Vers des solutions adaptatives et génériques pour l'extraction de motifs intéressants dans les données

Frédéric Flouvat

N° d'Ordre : D.U. 1710
EDSPIC : 364

Université Blaise Pascal - Clermont II

École doctorale
sciences pour l'ingénieur de Clermont-Ferrand

THÈSE

présentée par

Frédéric Flouvat

Formation doctorale :
Informatique, productique, imagerie médicale

pour obtenir le grade de

Docteur d'université
Spécialité : Informatique

Vers des solutions adaptatives et génériques pour l'extraction de motifs intéressants dans les données

Soutenue publiquement le 8 décembre 2006 devant le jury :

M. Alain Quilliot	Président
Mme Christine Collet	Rapporteur et examinateur
M. Pascal Poncelet	Rapporteur et examinateur
M. Michel Schneider	Examinateur
M. Farouk Toumani	Examinateur
M. Fabien De Marchi	Co-directeur de thèse
M. Jean-Marc Petit	Directeur de thèse

Remerciements

Je tiens tout d'abord à exprimer toute ma reconnaissance à mes directeurs de thèse Jean-Marc Petit et Fabien De Marchi pour leur encadrement, leurs nombreux conseils et leur soutien tout au long de ma thèse. Leur oeil critique m'a été très précieux, et m'a permis d'arriver là où j'en suis actuellement.

Je souhaite exprimer ma gratitude aux membres du jury pour avoir bien voulu consacrer une partie de leur temps à ma thèse. Je remercie plus particulièrement Christine Collet et Pascal Poncelet d'avoir accepté d'être mes rapporteurs. Leurs remarques et suggestions, tant sur le fond que sur la forme, ont été très pertinentes et appréciées.

J'adresse mes remerciements à Alain Quilliot et Michel Schneider pour m'avoir permis d'intégrer le LIMOS et son équipe base de données. Je remercie aussi Bernard Péroche et Robert Laurini pour m'avoir accueilli au LIRIS pendant mes derniers mois de thèse. Plus généralement, mes remerciements vont à l'ensemble des membres des laboratoires LIMOS et LIRIS. Je remercie plus particulièrement Marie et Hélène à Clermont-Ferrand, et Claudia, Reim, Sana à Lyon, pour leur amitié et pour avoir contribué à créer une ambiance agréable de travail.

Je tiens à remercier chaleureusement mes amis et ma famille pour m'avoir soutenu pendant ces années de thèse.

Enfin et surtout, merci de tout mon coeur à Sylvie pour ses attentions, son écoute et son aide de chaque instant.

"Vivez comme si vous deviez mourir demain,
apprenez comme si vous deviez vivre toujours."

Siddhārtha Gautama

Résumé

La découverte de motifs (ou *itemsets*) fréquents dans les données est un des problèmes de fouille de données les plus étudiés. Face à ce problème, un grand nombre d'algorithmes [AS94, ZPOL97, PBTL99, HPY00, BCG01, GZ01, UAUA03] ont été développés. Toutefois, pendant longtemps, aucune évaluation objective des algorithmes d'extraction de motifs fréquents n'avait été réalisée. Les ateliers FIMI (*Frequent Itemset Mining Implementation*) [BZ03, BGZ04] ont eu pour objectif de résoudre ce problème en étudiant ces algorithmes sur des jeux de données communs et en mettant à disposition de la communauté leur code source.

Afin de mieux comprendre l'influence des données sur les algorithmes, nous présentons une étude expérimentale des jeux de données communément utilisés par la communauté. Cette étude met notamment en avant l'influence des bordures des motifs fréquents sur les algorithmes, ainsi que la "stabilité" des distributions étudiées par rapport à la variation du seuil minimum de support. Ces résultats permettent d'aboutir à une nouvelle classification des jeux de données : stable et en accord avec les performances des algorithmes.

Face aux limites des approches existantes se focalisant uniquement sur la gestion des données, nous proposons par la suite un algorithme adaptatif de découverte des bordures des motifs fréquents, appelé *ABS* (*Adaptive Borders Search*). Cet algorithme, issu de l'étude expérimentale réalisée, adapte dynamiquement sa stratégie d'exploration de l'espace de recherche en fonction des bordures des motifs fréquents.

Malgré le grand nombre de travaux autour des motifs fréquents et le cadre théorique des problèmes d'extraction de motifs intéressants défini dans [MT97], l'utilisation des algorithmes de découverte des motifs fréquents pour résoudre d'autres problèmes "équivalents" est peu répandue et reste délicate. Dans ce contexte, un des intérêts d'*ABS* est de s'appuyer sur des stratégies génériques et indépendantes du problème, contrairement à la majeure partie des algorithmes d'extraction de motifs fréquents.

Afin de faciliter le développement de solutions logicielles pour l'extraction de motifs intéressants, nous proposons une librairie C++ basée sur l'utilisation d'algorithmes et de structures de données génériques, et passant à l'échelle. De plus, nous présentons une ébauche de méthodologie pour guider l'utilisateur dans ses choix. Cette librairie est directement issue de notre expérience dans le développement et l'adaptation d'algorithmes de découverte de motifs fréquents pour résoudre des problèmes divers telles que les représentations condensées des motifs fréquents, l'extraction de contraintes d'intégrité, et la résolution de certaines phases d'un processus de réécriture de requêtes.

Mots-clés : fouille de données, motifs fréquents, classification des jeux de données, algorithmes adaptatifs, extraction de motifs intéressants, librairie C++.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Première partie

Introduction

Contexte

Dans une grande variété de domaines, des données ont été collectées et accumulées à une très grande vitesse, et continuent de l'être. Face à ce déluge de données, le plus grand challenge est de fournir aux utilisateurs des techniques et outils permettant d'en extraire des informations utiles. De ce besoin est né un nouveau domaine de recherche à l'intersection de plusieurs disciplines telles que les bases de données, l'algorithmique, les statistiques, et l'intelligence artificielle : ce domaine est l'extraction de connaissances dans les données (ECD) [FPSS96]. Typiquement, l'ECD a pour objectif de transformer l'information contenue dans la grande quantité de données collectées sous une autre forme plus compacte, plus abstraite, et plus utile. L'ECD représente l'ensemble du processus de découverte de connaissance : de la gestion des données, à l'interprétation et la visualisation du résultat, en passant par la définition d'algorithmes pouvant traiter efficacement les grands volumes de données. La figure 1 représente en détail les différentes étapes de ce processus [FPSS96] :

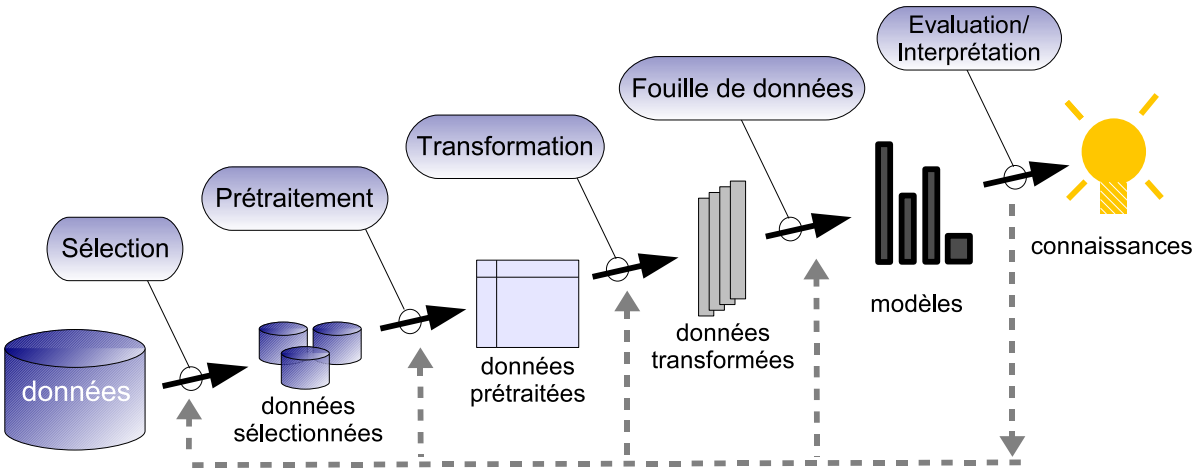

FIG. 1 – Processus d'extraction de connaissances dans les données

La première étape de ce processus est la compréhension du domaine d'application, des informations importantes, et des objectifs des utilisateurs finaux. Ensuite, il faut sélectionner les données à partir desquelles l'extraction de connaissances sera effectuée. Si nécessaire, ces données pourront être nettoyées et des prétraitements pourront être appliqués (suppression du bruit, gestion des données manquantes,...). Les données peuvent aussi être transformées par des réductions et projections. L'étape suivante est d'identifier la méthode de fouille de données correspondant aux types de connaissances recherchées par l'utilisateur, puis d'appliquer cette méthode sur les données traitées. Cette étape inclut aussi le choix des paramètres d'entrée de la méthode de fouille de données. Finalement, la dernière étape consiste à visualiser, interpréter et consolider la connaissance acquise. Ce processus est itératif, chaque étape peut entraîner des modifications au niveau des étapes précédentes.

Une grande partie des travaux en extraction de connaissances s'est focalisée sur la

fouille de données. La fouille de données est l'étape du processus d'extraction de connaissances qui consiste à appliquer des algorithmes d'analyse de données et de découverte produisant un ensemble de motifs (ou modèles) particuliers sur les données [FPSS96]. Cette discipline peut aussi être définie comme "l'analyse d'ensembles (souvent grands) de données issues d'observations pour trouver des relations inconnues et résumer les données de nouvelles manières, de façon à ce que les résultats soient compréhensibles et utiles pour le propriétaire des données" [HMS01].

La découverte de motifs (ou *patterns*) dans les données est un des problèmes de fouille de données les plus étudiés. L'objectif est de découvrir les motifs apparaissant "fréquemment" dans une base de données. Différents types de motifs ont été étudiés tels que des ensembles [AIS93], des séquences [AS95, MCP98], des graphes [IWM00], et des structures d'arbre [TRS02, Zak02]. Le problème ayant connu le plus grand nombre de contributions est sans aucun doute l'extraction d'ensembles d'articles, appelés aussi motifs (ou *itemsets*), fréquents dans une base de données de transactions telle que celle représentant les achats effectués par des clients dans un supermarché. Ces motifs sont notamment utilisés dans la découverte des règles d'association [AIS93].

L'extraction des motifs fréquents est une tâche difficile : la base de données peut contenir des millions de transactions et l'espace de recherche est exponentiel dans le nombre d'articles présents dans la base de données. Face à ce problème, un grand nombre d'algorithmes [AS94, ZPOL97, PBTL99, HPY00, BCG01, GZ01, UAUA03] ont été développés avec un objectif double : limiter l'espace de recherche étudié et optimiser l'accès aux données. Chaque nouvel algorithme était présenté comme étant plus performant que les algorithmes existants. Toutefois, ces résultats s'appuyaient sur des expérimentations réalisées sur un nombre très réduit de jeux de données et de seuils minimums de support. De plus, les implémentations originales des algorithmes étaient rarement disponibles ce qui rendait difficile une comparaison objective, d'autant que le comportement de différentes implémentations d'un même algorithme pouvait beaucoup varier.

Face à ces problèmes, les ateliers FIMI (*Frequent Itemset Mining Implementation*) [BZ03, BGZ04] ont eu pour but d'effectuer une évaluation objective des performances des algorithmes d'extraction de motifs fréquents, d'étudier le comportement de ces algorithmes sur les mêmes jeux de données et de mettre à disposition de la communauté leur code source. Plus récemment, l'atelier OSDM (*Open Source Data Mining*) [GNZ05] a eu ces mêmes objectifs en élargissant la portée aux algorithmes de "clustering" et à la classification.

Les jeux de données étudiés lors de ces ateliers sont des jeux de données réels et synthétiques couramment utilisés pour tester des algorithmes de découverte des motifs fréquents, motifs fréquents maximaux, ou des représentations condensées des motifs fréquents (le tableau 1 en présente quelques un). Par exemple, parmi les jeux de données réels, $Mushroom$ est une base de données qui recense près de 120 caractéristiques de certaines espèces de champignons, et $Pumsb$ contient des données issues du bureau de recensement américain. Deux jeux de données synthétiques sont aussi utilisés : $T10I4D100K$ et $T40I10D100K$.

Ces jeux de données ont été générés grâce au générateur d'IBM [QT] à partir de la méthode présentée dans [AS94]. Pour résumer, l'objectif de ce générateur est de créer des transactions similaires à celles obtenues dans un environnement de type supermarché. En appliquant certaines lois de distribution, le jeu de données tend à modéliser le monde réél par rapport à des critères donnés en entrée, tout en permettant l'existence d'exceptions.

Jeux de données	nb articles	taille moy. trans.	nb. trans.
Chess	75	37	3 196
Connect	129	43	67 557
Mushroom	119	23	8 124
Pumsb*	2 088	50.5	49 046
Pumsb	2 113	74	49 046
T10I4D100K	1000	10	100 000
T40I10D100K	1000	40	100 000

TAB. 1 – Caractéristiques de jeux de données étudiés lors de FIMI

Les expérimentations réalisées sur ces jeux de données ont comparé les performances des algorithmes en terme d'occupation mémoire et de temps d'exécution en faisant varier le seuil minimum de support. Au cours des deux ateliers FIMI, près de vingt-cinq implémentations d'algorithmes et versions d'algorithmes ont été testées, avec pour objectif la découverte des motifs fréquents et/ou des motifs fréquents maximaux et/ou des motifs fermés fréquents. Parmi les algorithmes étudiés figurent des algorithmes classiques tels qu'*Apriori* [AS94] et *Eclat* [ZPOL97], mais aussi les derniers algorithmes développés tels que $Mafia$ [BCG01], *GenMax* [GZ01], $FPmax*$ [GZ03], *Dualize and Advance* [GKM+03], ou LCM [UAUA03, UKA04].

En parallèle de ces travaux, de plus en plus de problèmes ont été identifiés comme équivalents à un problème d'extraction de motifs fréquents [MT97, GKM+03]. Ces problèmes abordent des domaines variés tels que la découverte de contraintes d'intégrité dans des bases de données relationnelles [MT97, MLP02] ou la résolution de certaines phases d'un processus de réécriture de requêtes [JPR+05]. L'ensemble de ces problèmes peut donc en théorie profiter de la grande quantité de travail issue de l'extraction de motifs fréquents, et plus particulièrement des différentes stratégies pour explorer l'espace de recherche efficacement. Dans [MT97], un cadre théorique unique a même été proposé pour représenter l'ensemble de ces problèmes, et les reformuler en problème d'extraction de motifs (au sens large) intéressants dans les bases de données.

Problèmes et motivations

Adaptation des algorithmes d'extraction de motifs fréquents aux données

Les caractéristiques des données ont naturellement un fort impact sur les algorithmes. Il est donc important de développer des algorithmes adaptatifs, i.e. adaptant dynamiquement leur stratégie en fonction des données. Même si la majorité des algorithmes intègrent des aspects adaptatifs, la stratégie globale de l'algorithme s'adapte peu aux données. Ces aspects se focalisent sur la gestion des données, et non sur la stratégie même de parcours de l'espace de recherche. Or, les résultats des expérimentations réalisées lors de FIMI ont fait apparaître que ces stratégies ne suffisent pas face à la grande diversité des données.

Dans ce contexte, la mise en place d'algorithmes adaptatifs modifiant *dynamiquement* leur stratégie d'exploration de l'espace de recherche en fonction des données rencontrées est une perspective particulièrement intéressante. Toutefois, le développement de tels algorithmes nécessite une bonne maîtrise à la fois des stratégies existantes et de la manière dont elles sont influencées par les données, ce qui n'est pas encore totalement le cas. En effet, certains résultats obtenus lors de FIMI ne furent pas expliqués. Pour deux jeux de données en apparence comparables, un algorithme pouvait avoir un comportement totalement différent, et ceci indépendamment du seuil minimum de support. Ces constatations découlent de l'observation des principales caractéristiques identifiées des jeux de données : le nombre de transactions de la base de données, leur taille et la *densité* [GZ01]. Un jeu de données est dit "dense" lorsqu'il produit de longs motifs fréquents même pour des valeurs élevées de seuil minimum de support [GZ01]. Ces caractéristiques des données ne suffisent donc pas pour refléter les informations contenues dans les données et expliquer le comportement des algorithmes.

Utilisation des algorithmes d'extraction de motifs fréquents pour résoudre des problèmes équivalents

Malgré le grand nombre de travaux autour des motifs fréquents et le cadre théorique des problèmes d'extraction de motifs intéressants défini dans [MT97], l'utilisation des algorithmes de découverte des motifs fréquents pour résoudre d'autres problèmes "équivalents" est peu répandue et reste délicate. Les deux raisons suivantes permettent d'expliquer cette constatation.

Tout d'abord, les algorithmes développés dans le cadre de la découverte des motifs fréquents s'appuient sur des propriétés et des techniques spécifiques à ce problème. Par exemple, des algorithmes tels que $FP-growth$ [HPY00] ou LCM [UAUA03] peuvent difficilement être appliqués à un autre problème. En général, seule une partie des algorithmes peut être réutilisée : la stratégie d'exploration de l'espace de recherche. Or, l'objectif prin-

cipal de ces algorithmes n'est pas de limiter l'espace de recherche parcouru, mais surtout d'optimiser le test des motifs grâce à certaines propriétés des motifs fréquents. En effet, les algorithmes les plus performants ont principalement optimisé le test des motifs au détriment de la généricité.

La mise en place de ces algorithmes au niveau logiciel est longue et complexe. Afin de garantir leur efficacité, une attention particulière doit être portée aux structures de données et à l'implémentation de l'algorithme. Le développement de ce type de programme nécessite donc de fortes compétences en développement logiciel et une très bonne maîtrise des algorithmes. Par exemple, l'implémentation de l'algorithme *Apriori* de C. Borgelt [Bor03] (la plus performante actuellement) a été proposée en 2002 et est encore maintenant mise à jour. Autre exemple, l'implémentation de l'algorithme *LCM* [UAUA03] présentée lors de FIMI 2003, et élue meilleure implémentation de FIMI 2004, en est à sa troisième version actuellement.

En dépit du grand nombre d'implémentations et de codes sources d'algorithmes d'extraction de motifs fréquents librement accessibles sur internet [BZ03, BGZ04, GNZ05], l'adaptation de ces programmes pour résoudre tout autre problème que le problème initialement étudié s'avère aussi difficile : même une légère modification de leur code peut être insurmontable. En effet, toutes ces implémentations ont été développées et optimisées, bien souvent de façon brillante et astucieuse d'un point de vue technique. Le corollaire est que le code source exige une étude approfondie sur le plan de la gestion de la mémoire et de l'accès aux données en particulier, pour espérer être adapté. A notre connaissance, la seule contribution visant à simplifier la réutilisation de ces algorithmes est la librairie *DMTL* [HCS+05]. Toutefois, elle se focalise sur les problèmes d'extraction de motifs fréquents (au sens large), et ne peut être appliquée aux autres problèmes d'extraction de motifs intéressants.

Contribution du mémoire

Dans ce contexte, les trois principales contributions de notre travail ont été les suivantes :

- L'identification de nouvelles caractéristiques importantes des données, et la proposition d'une nouvelle classification des jeux de données [FMP05]. Pour faire cela, nous avons conduit une étude expérimentale poussée des jeux de données communément utilisés par la communauté.
- Le développement d'un algorithme adaptant dynamiquement sa stratégie d'exploration de l'espace de recherche en fonction des données, appelé *ABS* (*Adaptive Borders Search*) [FMP04]. Les aspects adaptatifs de cet algorithme sont directement issus de l'étude expérimentale réalisée.
- L'étude des algorithmes d'extraction de motifs fréquents pour résoudre des problèmes équivalents. Plus précisément, nous avons proposé une solution logicielle [FMP06] pour faciliter l'utilisation de ces algorithmes pour résoudre de manière générique tout problème de découverte de motifs intéressants dans les données,

correspondant au cadre théorique de [MT97].

Une nouvelle caractérisation des jeux de données Nous avons effectué une étude expérimentale poussée des jeux de données afin d'identifier de nouvelles caractéristiques importantes [FMP05]. Lors de ces expérimentations, nous avons étudié quatre types de motifs : les motifs fréquents, fermés fréquents [PBTL99], générateurs fréquents [PBTL99, BBR03], et essentiels fréquents [CCL05]. De plus, lorsque cela était possible, les bordures positive et négative de ces motifs ont aussi été recherchées. La nouveauté réside dans l'étude de la bordure négative qui, en complément de la bordure positive, permet d'avoir une caractérisation plus fine des jeux de données. Cette étude a pour objectif une meilleure maîtrise des algorithmes existants et une meilleure compréhension de l'influence des données sur ceux-ci, dans le but notamment de pouvoir développer des algorithmes adaptatifs.

Au final, ces expérimentations ont mis en avant l'influence des bordures positive et négative des motifs fréquents sur les algorithmes. L'étude de la "distance" entre les bordures des motifs fréquents donne des informations importantes sur les performances de différents types d'algorithmes. De plus, elle a fait apparaître différentes propriétés non justifiées d'un point de vue théorique, telle que la *"stabilité"* des distributions étudiées par rapport à la variation du seuil minimum de support.

A partir de ces résultats, une nouvelle classification des jeux de données a été proposée. Elle diffère de celle proposée dans [GZ01] car elle prend en compte la bordure négative des motifs fréquents en plus de la bordure positive et s'appuie sur la "distance" entre ces deux bordures pour classer les jeux de données. La classification proposée a les avantages suivants :

- une meilleure correspondance entre la classification et les performances des algorithmes. En d'autres termes, cette classification est une première étape vers l'évaluation de la "difficulté" d'un jeu de données ;
- la stabilité de la classification par rapport au changement de seuil minimum de support.

Mise en place d'un algorithme adaptatif de découverte des bordures des motifs fréquents A partir de l'étude expérimentale réalisée, et soulignant l'influence de la bordure négative sur les performances des algorithmes, nous proposons un algorithme adaptatif, appelé *ABS* (*Adaptive Borders Search*) [FMP04], permettant de découvrir les bordures positive et négative des motifs fréquents. Cet algorithme est issu des précédents travaux de l'équipe pour l'algorithme *ZigZag* [MP03] et l'extraction des dépendances d'inclusions les plus spécialisées. *ABS* s'appuie sur deux stratégies d'exploration de l'espace de recherche :

- La première consiste à utiliser l'algorithme *Apriori* [AS94] pour explorer l'espace de recherche par niveau des motifs les plus petits aux plus grands.
- La deuxième alterne des "sauts" ou *dualisations* [MT97, MP03] entre les deux bordures en construction. Les sauts effectués exploitent l'anti-monotonie du prédicat pour éviter de tester un grand nombre de motifs.

ABS est à notre connaissance la première approche à adapter *dynamiquement* sa stratégie de parcours de l'espace de recherche. Cette stratégie adaptative consiste dans un premier temps à déterminer dynamiquement à quel moment l'algorithme doit passer d'une exploration par niveau, à une approche effectuant des "sauts" dans l'espace de recherche. Ce choix se fait principalement en fonction de la bordure négative des motifs fréquents en construction. Dans un second temps, la stratégie adaptative exploite une mesure d'erreur pour identifier les motifs "presque vrais" découverts lors des sauts, et les utilise pour explorer différemment certaines parties de l'espace de recherche. Les résultats expérimentaux obtenus montrent l'intérêt de changer dynamiquement la stratégie d'exploration de l'espace de recherche en fonction des données.

De plus, un des intérêts d'ABS est de s'appuyer sur des stratégies génériques et indépendantes du problème, contrairement à la majeure partie des algorithmes d'extraction de motifs fréquents. Même sa stratégie adaptive peut être généralisée au cadre des motifs intéressants de [MT97]. Cette remarque nous conduit à la troisième contribution de ce mémoire.

Vers une solution logicielle pour les problèmes d'extraction de motifs intéressants Afin de faciliter le développement de solutions logicielles pour l'extraction de motifs intéressants, nous proposons une librairie basée sur l'utilisation d'algorithmes et de structures de données génériques, et passant à l'échelle. Les caractéristiques et optimisations de l'algorithme sont transparentes pour le programmeur, et seule l'implémentation des propriétés spécifiques à son problème est laissée à sa charge. De plus, nous présentons une méthodologie pour guider l'utilisateur dans ses choix. Cette méthodologie l'aide à reformuler (si possible) son problème sous la forme d'un problème d'extraction de motifs intéressants, et l'assiste aussi dans le développement des différents composants.

Cette librairie est directement issue de notre expérience dans le développement et l'adaptation d'algorithmes de découverte de motifs fréquents pour résoudre des problèmes telles que : la découverte de représentations condensées des motifs fréquents (générateurs fréquents, essentiels fréquents) [FMP05], l'extraction de dépendances d'inclusion [MP03] et la résolution de certaines phases du processus de réécriture de requêtes dans un système d'intégration de sources de données hétérogènes [JF06]. La principale nouveauté de cette contribution est d'aborder le problème de la généricité et du temps de développement nécessaire pour obtenir des programmes fiables, robustes et passant à l'échelle, pour tout problème correspondant au cadre théorique de [MT97]. Contrairement aux autres contributions, ce travail permet donc de résoudre une grande famille de problèmes.

Les premières expérimentations réalisées affichent des performances très intéressantes au regard de la rapidité et la simplicité d'obtention d'un programme opérationnel. La librairie en cours de développement, implantée en C++, est accessible sur internet et libre de droits.

Organisation du mémoire

Ce mémoire fort de cinq parties est organisé de la façon suivante.

La partie suivante introduit un ensemble de définitions et de propriétés autour des motifs fréquents et d'autres représentations condensées des motifs fréquents (chapitre 1). Le cadre théorique de découverte de connaissances de [MT97] permettant de formuler de manière générique les problèmes d'extraction de motifs (au sens large) intéressants est présenté au chapitre 3. Les principales stratégies d'exploration de l'espace de recherche pour cette famille de problème sont ensuite détaillées (chapitre 4).

La troisième partie se focalise sur la mise en place d'un algorithme adaptatif pour l'extraction des motifs fréquents. Le chapitre 5 dresse un état de l'art des principales caractéristiques des jeux de données connues, et souligne leurs limites, notamment par rapport aux performances des algorithmes (section 5.1). La section 5.2 de ce chapitre présente les principales stratégies adaptatives mises en place dans les algorithmes de découverte des motifs fréquents. Ensuite, le chapitre 6 traite du problème de la caractérisation des données. L'étude expérimentale effectuée et la classification des jeux de données des motifs fréquents qui en découlent sont présentées dans ce chapitre. Les résultats obtenus lors de l'étude expérimentale sont détaillés dans la section 6.1. Ensuite, la section 6.2 étudie l'impact sur les algorithmes de certaines caractéristiques identifiées lors de ces expérimentations. Finalement, la section 6.3 présente comment les résultats obtenus permettent d'aboutir à une nouvelle classification des jeux de données et des problèmes. Le chapitre 7 introduit l'algorithme ABS. Le principe de cet algorithme est présenté dans la section 7.1, avec une attention plus particulière sur les aspects adaptatifs et la nouvelle génération des candidats. L'algorithme est ensuite décrit dans son ensemble en section 7.2. Finalement, les sections 7.3 et 7.4 présentent les expérimentations réalisées et discutent les résultats obtenus.

La quatrième partie traite du problème de l'utilisation et de l'implémentation des algorithmes d'extraction de motifs fréquents pour résoudre des problèmes d'extraction de motifs (au sens large) intéressants respectant le cadre de [MT97]. Le chapitre 8 étudie comment se comporte les solutions algorithmiques et logicielles existantes pour les motifs fréquents dans ce nouveau contexte. La section 8.2 met en évidence les limites de la majeure partie des algorithmes d'extraction de motifs fréquents, et souligne l'intérêt de l'algorithme ABS. La section 8.3 présente les principales solutions logicielles existantes. Le chapitre 9 présente notre librairie facilitant le prototypage rapide de ce type d'algorithmes. Le cadre théorique de cette librairie est abordé en section 9.1. La section 9.2 décrit une méthodologie permettant de vérifier qu'un problème correspond à ce cadre. Ensuite, l'organisation et le fonctionnement de la librairie sont présentés dans la section 9.3. Quelques résultats expérimentaux sont introduits en section 9.4.

Finalement, nous concluerons et donnerons quelques perspectives à ce travail dans la dernière partie de ce mémoire.

Deuxième partie

Extraction de motifs

Chapitre 1

Les motifs fréquents

Le problème de la découverte des motifs fréquents a été introduit la première fois par R. Agrawal et Al. [AIS93] dans le cadre de la découverte des règles d'association. Cette nouvelle notion est à l'origine liée au besoin d'analyser des données de type "transactions d'un supermarché" afin d'en déduire le comportement d'achat des clients. Depuis, ce problème joue un rôle important dans beaucoup de tâches et de problèmes de fouille de données telles que l'étude de séquences [AS95] ou la classification [LHM98]. Dans ce chapitre, nous allons introduire les principales notions liées aux motifs fréquents.

Définitions

Soit R un ensemble fini de symboles appelés *articles*. Une *transaction* est un sous-ensemble de R . Une *base de données de transactions* r sur R est un multi-ensemble de transactions.

Définition 1:
Un ensemble $X = \{i_1, ..., i_k\} \subseteq R$ est appelé un motif ou k-motif s'il contient k articles.

Nous notons $cover(r, X) = \{t \in r \mid X \subseteq t\}$ le multi-ensemble de transactions contenant le motif X.

Définition 2:
Le support d'un motif X dans R représente le nombre de transactions qui contiennent X :

$$support(r, X) = |cover(r, X)|$$

Un motif est dit *fréquent* si son support est supérieur à un seuil minimum de support absolu $minsup$, où $minsup \in \{0, ..., |r|\}$. Notons que le support d'un motif et le seuil

minimun de support peuvent être aussi définis de manière relative, i.e. dans $[0, 1]$.

Définition 3:
L'ensemble de tous les motifs minsup -fréquents dans r est :

$$F = \{X \subseteq R \mid support(r, X) \geq minsup\}$$

L'ensemble des motifs fréquents de taille k sera noté F_k . La propriété "être motif fréquent dans une base de données r " est anti-monotone relativement à la relation "$\subseteq$", comme indiqué dans la propriété suivante.

Propriété 1:
Soit $X, Y \subseteq R$ avec $X \subseteq Y$. Y est fréquent $\Rightarrow$ X est fréquent.

Illustrons maintenant ces définitions par un exemple classique représentant les achats effectués par des clients dans un magasin d'alimentation.

Exemple 1:
Soit la base de données de transactions r (tableau 1.1), où chaque transaction représente les articles achetés par un client. L'ensemble des articles est :

$$R = \{pain,\ lait,\ bières,\ couches\}$$

T1	{pain }
T2	{ lait, bières, couches}
T3	{ bières, couches }
T4	{pain, lait, bières, couches }
T5	{ bières, couches }

TAB. 1.1 – Exemple de base de données de transactions r

L'ensemble des motifs de r ainsi que leur support est donné dans le tableau 1.2.

Si le seuil minimum de support est de 2, l'ensemble des motifs fréquents est donc :

F = { { $\emptyset$ } , {pain}, {lait}, {bières}, {couches}, {lait,bières}, {lait,couches}, {bières,couches}, {lait,bières,couches} }

L'espace de recherche

L'extraction des motifs fréquents est une tâche difficile : la base de données peut contenir des millions de transactions et l'espace de recherche est exponentiel dans le

Motif	Support		Motif	Support
{∅}	5		{lait,bières}	2
{pain}	2		{lait,couches}	2
{lait}	2		{bières,couches}	4
{bières}	4		{pain,lait,bières}	1
{couches}	4		{pain,lait,couches}	1
{pain,lait}	1		{pain,bières,couches}	1
{pain,bières}	1		{lait,bières,couches}	2
{pain,couches}	1		{pain,lait,bières,couches}	1

TAB. 1.2 – Motifs et leur support dans r

nombre d'articles présents dans la base de données.

L'espace de recherche des motifs peut être représenté par le treillis des parties de R , i.e. composé de $2^{|R|}$ motifs. La figure 1.1 montre l'espace de recherche de l'exemple 1.

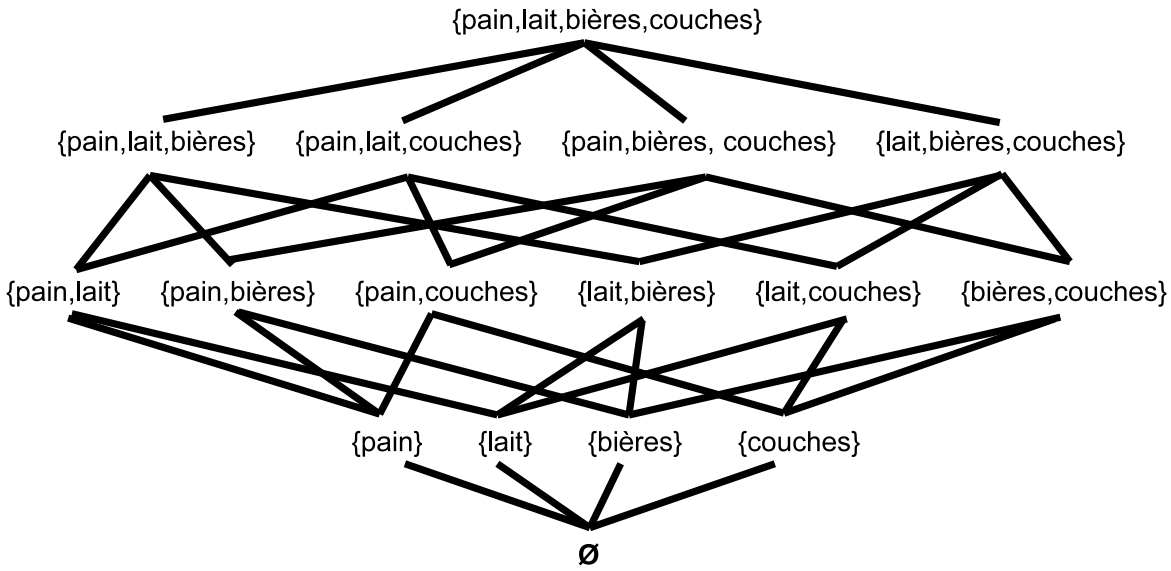

FIG. 1.1 – Exemple de treillis des motifs

Par conséquent, lorsque R est grand, l'espace de recherche devient trop important pour pouvoir générer et compter le support de tous les motifs.

Dans la suite du manuscrit, nous appelerons *niveau de l'espace de recherche* l'ensemble des motifs de même taille. Par exemple, l'exploration du k-ième niveau de l'espace de recherche représentera l'étude de l'ensemble des motifs de taille k.

Les bordures

La notion de bordures [MT97] est couramment étudiée pour les motifs fréquents. Il existe deux bordures : la bordure positive $\mathcal{B}d^+(F)$ et la bordure négative $\mathcal{B}d^-(F)$. Intuitivement, la bordure positive de l'ensemble des motifs fréquents F est constituée des éléments maximaux par inclusion de F ; sa bordure négative est constituée des éléments minimaux par inclusion n'appartenant pas à F . On a donc :

Définition 4:

$$\mathcal{B}d^+(F) = \{X \in F \mid \forall Y \subset X,\ Y \notin F\}$$
$$\mathcal{B}d^-(F) = \left\{X \in 2^R \backslash F \mid \forall Y \subset X, Y \in F\right\}$$

La figure 1.2 représente les bordures des motifs fréquents pour l'exemple 1 avec un seuil minimum de support de 2 (les nombres sous les motifs sont les supports).

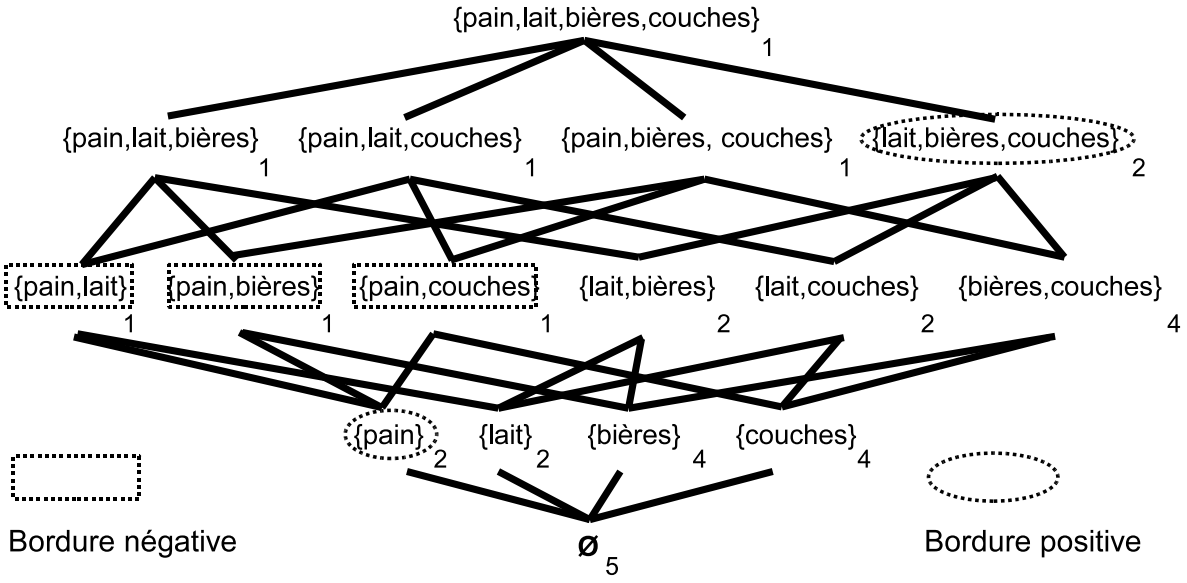

FIG. 1.2 – Exemple de bordures des motifs fréquents

Les bordures positive et négative des motifs fréquents permettent chacune de représenter l'ensemble des motifs fréquents F . En effet, de par l'anti-monotonie de la propriété "être un motif fréquent", nous avons les deux propriétés suivantes :

Propriété 2:

$$X \in F \Leftrightarrow \exists Z \in \mathcal{B}d^+(F) \mid X \subseteq Z$$
$$X \in F \Leftrightarrow \nexists Y \in \mathcal{B}d^-(F) \mid Y \subseteq X$$

Toutefois, ces bordures ne permettent pas de retrouver le support des motifs mais seulement de déterminer si un motif est fréquent ou non.

Chapitre 2

Représentations condensées des motifs fréquents

Parallèlement aux motifs fréquents, des représentations *condensées* ont été étudiées [PBTL99, BBR03, CCL05]. Leur objectif est double : améliorer l'efficacité de l'extraction des motifs fréquents quand cela est possible, et compresser le stockage de ces motifs pour un usage ultérieur.

De manière formelle, une représentation est dite condensée si elle est équivalente aux motifs fréquents : il est possible de retrouver chaque motif fréquent *ainsi que son support* sans accéder aux données [CG03].

La représentation condensée la plus étudiée est l'ensemble des *motifs fermés fréquents* [PBTL99, PHM00, ZH02]. Il existe aussi d'autres représentations condensées des motifs fréquents telles que : les motifs générateurs fréquents [PBTL99, BBR03] et les motifs essentiels fréquents [CCL05]. Notons que ces deux derniers ensembles ne sont pas totalement suffisants pour représenter l'ensemble des fréquents, puisqu'ils ont besoin des motifs d'une des deux bordures pour devenir une représentation condensée.

Nous avons choisi d'introduire ces motifs car ils couvrent les principaux types de représentations existantes et reflètent des informations importantes à propos des données.

Les fermés

Cette représentation est issue de la notion de *fermeture* utilisée en analyse formelle de concept [Wil82, GW99]. Elle a été initialement appliquée au problème de la découverte des motifs fréquents par l'équipe de Lotfi Lakhal dans [PBTL99].

Soit un motif X, la *fermeture* de X, notée $Cl(X)$, est l'ensemble des articles qui

apparaissent dans toutes les transactions où X apparaît. Formellement, soit une base de données de transactions r et un motif X :

Définition 5:

$$Cl(X) = \bigcap_{t \in r | X \subseteq t} \{t\}$$

Un motif X est *fermé* si $Cl(X) = X$.

Autrement dit, X est fermé dans r si et seulement si aucun sur-ensemble de X n'a le même support que X dans r , i.e. :

Propriété 3:

$$X \text{ fermé } \Leftrightarrow \forall Y \supset X, support(r, X) > support(r, Y)$$

Notons *Closed* l'ensemble des motifs fermés de r . La figure suivante représente l'ensemble des motifs fermés pour l'exemple 1.

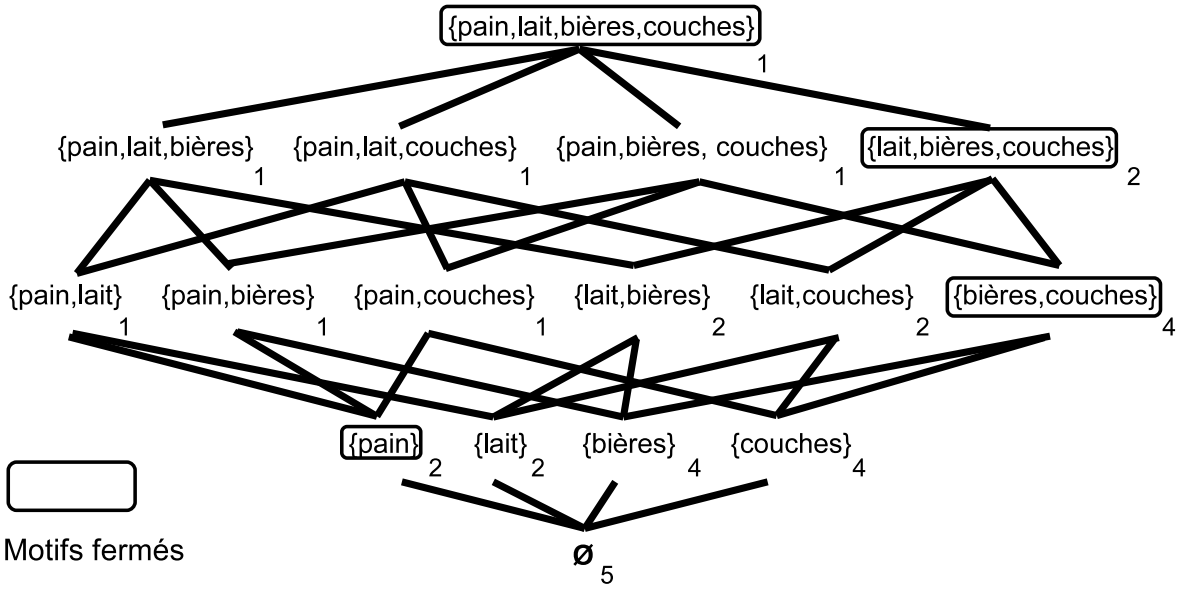

FIG. 2.1 – Exemple de motifs fermés

La bordure positive des motifs fréquents $\mathcal{B}d^+(F)$ contient donc par définition des motifs fermés. Plus précisément, l'ensemble des motifs fermés fréquents maximaux par inclusion forme la bordure positive.

De plus, d'après la définition de la fermeture d'un motif, on a la propriété suivante :

Propriété 4:

$$support(r, X) = support(r, Cl(X))$$

D'un point de vue plus pratique, le support d'un motif (qui n'est pas fermé) est égal au support du plus petit fermé qui le contient.

Pour un seuil minimum de support donné, il suffit donc de connaître l'ensemble de motifs fermés fréquents (noté $FClosed$) pour pouvoir générer tous les motifs fréquents et leur support.

Exemple 2:
Reprenons l'exemple de la figure 2.1, pour un seuil minimum de support de 2, l'ensemble des motifs fermés fréquents FClosed est { {pain}, {bières,couches}, {lait,bières,couches}}. A partir de ces motifs et de leur support, il est possible de déterminer si un motif est fréquent et la valeur de son support.
Etudions par exemple les motifs {pain, lait} et {lait,bières}. Le motif {pain, lait} est non fréquent car $\{pain, lait\} \not\subseteq X, X \in FClosed$. *Le motif {lait,bières} est fréquent car il est inclus dans le motif {lait,bières,couches}* $\in FClosed$. *Son support est égal au support de son plus petit sur-ensemble dans FClosed , i.e. {lait,bières,couches}. Son support est donc égal à 2.*

Les générateurs

La notion de *générateur* (appelé *libre* dans [BBR03]) a été introduite dans [PBTL99]. Cette notion est directement liée à la notion de fermé.

Soient X et Y deux motifs de l'espace de recherche. Définissons θ une relation binaire sur 2^R tel que $X\ \theta\ Y \Leftrightarrow cover(r, X) = cover(r, Y)$, i.e. X et Y sont dans les mêmes transactions. θ est une relation d'équivalence.

Les motifs peuvent alors être regroupés en *classes d'équivalence* par rapport à θ. Plus formellement, la classe d'équivalence contenant un motif X peut être définie de la manière suivante :

Définition 6:

$$[X] = \{Y \subseteq R \mid X\ \theta\ Y\}$$

A partir de cette définition, on obtient donc les propriétés suivantes :

Propriété 5:

$$X\ \theta\ Y \Rightarrow support(r, X) = support(r, Y)$$
$$X \subseteq Y\ \ et\ support(r, X) = support(r, Y) \Rightarrow X\ \theta\ Y$$

Par définition, une classe d'équivalence est donc composée de l'ensemble des motifs ayant la même fermeture.

Dans ce contexte, un motif X est *générateur* si X appartient aux plus petits motifs par rapport à l'inclusion de sa classe d'équivalence.

Définition 7:

$$X\ est\ un\ générateur\ si\ X \in \min_{\subseteq}[X]$$

Les motifs générateurs peuvent être détectés efficacement grâce à la propriété suivante :

Propriété 6:

$$X\ est\ générateur \iff \forall i \in X, support(r, X) < support(r, X - i)$$

La figure 2.2 représente l'ensemble des générateurs pour l'exemple 1. Les motifs fermés apparaissent sur ce schéma à titre d'indication. Les classes d'équivalence sont représentées par des couleurs différentes.

Notons Gen l'ensemble des motifs générateurs de r . Le support d'un motif X peut être déterminé à partir de Gen de la manière suivante :

Propriété 7:

$$support(r, X) = \min_{\forall Y \in Gen, Y \subseteq X}(support(r, Y))$$

Pour un seuil minimum de support donné, les motifs générateurs fréquents permettent donc de retrouver le support d'un motif fréquent. Toutefois, ils ne suffisent pas pour déterminer si un motif quelconque est fréquent ou pas. Pour cela, il est nécessaire de conserver en plus la bordure positive ou la bordure négative.

Notons que chaque motif fermé est donc l'élément maximal d'une classe d'équivalence. Les motifs générateurs étant les plus petits éléments des classes d'équivalence, leur nombre est donc nécessairement plus grand ou égal au nombre de motifs fermés. Par extension, il

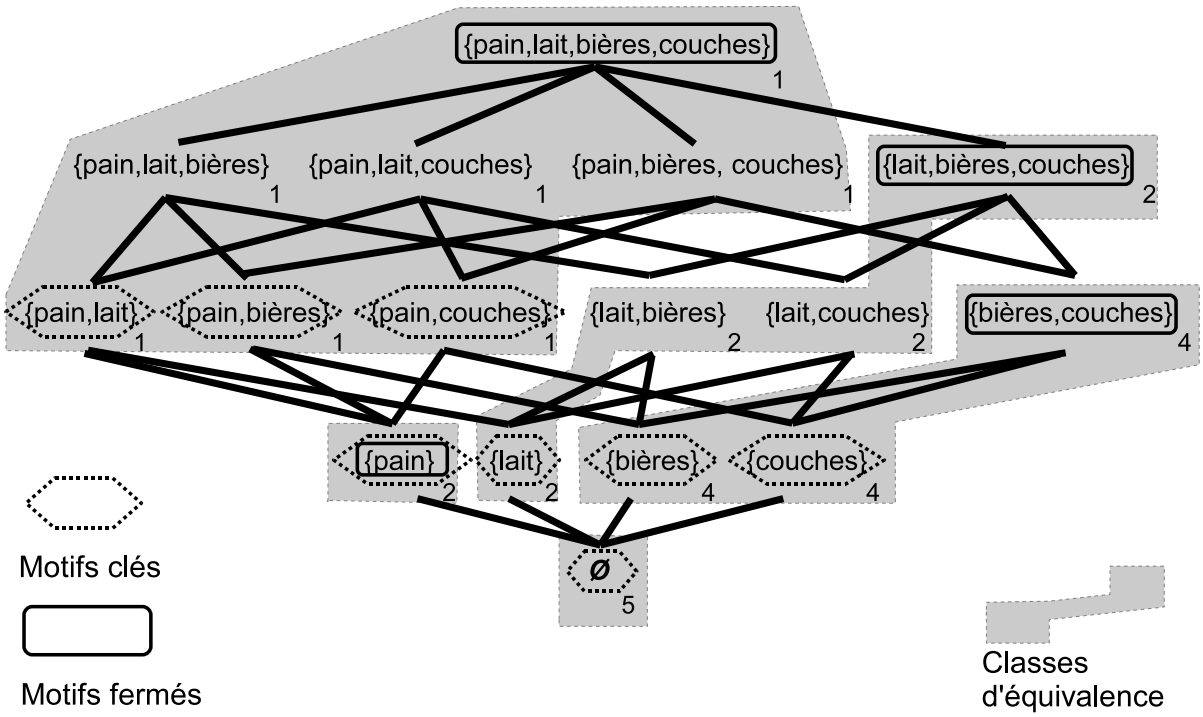

FIG. 2.2 – Exemple de motifs générateurs

en est de même pour les motifs générateurs fréquents et les motifs fermés fréquents pour un seuil minimum de support donné.

Les essentiels

La notion de motif essentiel a été introduite récemment dans [CCL05]. Elle s'appuie sur la notion de règle disjonctive [BR01, KG02]. Par ailleurs, d'autres représentations basées sur la notion de règle disjonctive ont été définies et présentées dans le cadre général proposé dans [CG03].

Une *règle disjonctive* est de la forme $X \rightarrow A_1 \vee A_2 ... \vee A_n$. Une telle règle est satisfaite si chaque transaction contenant X contient au moins un des éléments $A_1, ..., A_n$.

Définition 8:
Un motif X est dit essentiel s'il n'y a pas de règle disjonctive de la forme $A_1 \rightarrow A_2 \vee \ldots \vee A_k$, où $(A_i)_{i=1..k}$ sont des éléments distincts dans X.

Tout comme les motifs générateurs, ils peuvent être efficacement testés en exploitant la propriété suivante :

Propriété 8:

$$X \text{ est un motif essentiel} \iff \forall x \in X, disj(r, X) > disj(r, X - x)$$
$$\text{où } disj(r, X) = |\{t \in r \mid t \cap X \neq \emptyset\}|$$

L'ensemble des motifs essentiels de r pour l'exemple 1 est donné dans le tableau 2.1.

Motif essentiel X	$disj(r, X)$		Motif essentiel X	$disj(r, X)$
{∅}	5		{couches}	4
{pain}	2		{pain,bières}	5
{lait}	2		{pain,couches}	5
{bières}	4		{pain,lait}	3

TAB. 2.1 – Motifs essentiels de l'exemple 1

Pour un seuil minimum de support donné, l'ensemble des motifs essentiels fréquents n'est pas suffisant pour déterminer si un motif est fréquent ou non. Pour obtenir une représentation condensée, il faut conserver aussi une des deux bordures.

Chapitre 3

Généralisation : un cadre théorique de découverte de connaissances

Un grand nombre de problèmes sont équivalents au problème de l'extraction des motifs fréquents [MT96]. Nous présentons dans ce chapitre le cadre théorique défini dans [MT97] et généralisant les problèmes de découverte de motifs intéressants (motifs au sens large, pas uniquement des ensembles).

Soit r une base de données, un langage fini $\mathcal{L}$ pour exprimer des motifs (ou "mots") ou définir des sous-groupes des données, et un prédicat Q permettant d'évaluer si un motif $\varphi \in {}^1\mathcal{L}$ est vrai, ou "intéressant" dans r .

Soit la tâche de fouille de données consistant à extraire les motifs intéressants de r relativement à $\mathcal{L}$ et Q , appelée théorie de $\mathcal{L}$, r , Q dans [MT97], et définie par :

$$Th(\mathcal{L}, r, Q) = \{\varphi \in \mathcal{L} | Q(r, \varphi) \text{ est vraie}\}$$

Supposons en outre qu'une *relation de spécialisation/généralisation*, notée $\preceq$ est définie sur les éléments de $\mathcal{L}$. On dira que φ est plus général (resp. plus spécifique) que θ, si $\varphi \preceq \theta$ (resp. $\theta \preceq \varphi$). Si le prédicat Q est monotone (resp. anti-monotone) par rapport à l'ordre $\preceq$, alors pour chaque $\theta, \varphi \in \mathcal{L}$ tels que $\varphi \preceq \theta$, on a : $Q(r, \varphi)$ est vrai (resp. faux) $\Longrightarrow Q(r, \theta)$ est vrai (resp. faux).

Notons $rank(\varphi)$le *rang* d'un motif $\varphi \in \mathcal{L}$. S'il n'existe pas de $\theta \in \mathcal{L}$ tel que $\theta \preceq \varphi$, alors $rank(\varphi) = 0$, sinon $rank(\varphi) = 1 + max\{rank(\theta) \mid \theta \prec \varphi\}$. Pour $T \subset \mathcal{L}$, soit T_i l'ensemble des motifs de $\mathcal{L}$ de rang i.

Lorsque le prédicat est anti-monotone, l'ensemble $Th(\mathcal{L}, r, Q)$ est dit "fermé par le bas". Dans ce cas, les notions de bordures introduites pour les motifs fréquents en section 1 peuvent être généralisées à ce cadre théorique :

[1] symbole utilisé dans [MT97] pour représenter l'appartenance d'un motif au langage.

- *Sa bordure positive* est composée des motifs intéressants les plus spécialisés par rapport à la relation d'ordre, définie par

$$\mathcal{B}d^+(Th(\mathcal{L},r,Q)) = \{\sigma \in Th(\mathcal{L},r,Q) \mid \nexists \varphi \in Th(\mathcal{L},r,Q), \sigma \prec \varphi\}$$

- *Sa bordure négative* est composée des motifs non intéressants les plus généraux par rapport à la relation d'ordre, définie par

$$\mathcal{B}d^-(Th(\mathcal{L},r,Q)) = \{\sigma \in \mathcal{L} \setminus Th(\mathcal{L},r,Q) \mid \nexists \varphi \in \mathcal{L} \setminus Th(\mathcal{L},r,Q), \varphi \prec \sigma\}$$

Tout comme pour les motifs fréquents, la théorie de $\mathcal{L}$ peut alors être représentée de façon équivalente par ses bordures positive et négative.

Si le prédicat est monotone, l'ensemble $Th(\mathcal{L},r,Q)$ est dit "fermé par le haut", et la notion de bordures existe aussi. La bordure positive est alors composée des motifs intéressants les plus généraux, et la bordure négative des motifs non intéressants les plus spécialisés.

L'union de ces deux bordures positive et négative constitue *la bordure* de $Th(\mathcal{L},r,Q)$.

Notons que lorsqu'une mesure est associée aux motifs intéressants, les bordures peuvent ne pas suffire pour "reconstruire" l'ensemble des motifs et leur mesure associée, comme nous l'avons vu pour les motifs fréquents et le *support*.

Soit $(\mathcal{L},\preceq)$ l'ensemble ordonné composé des mots du langage $\mathcal{L}$, et d'un ordre partiel (relation de généralisation) $\preceq$ sur $\mathcal{L}$. Soit également R un ensemble fini d'éléments (ou atomes). Il est parfois possible de définir un isomorphisme f de $(\mathcal{L},\preceq)$ vers $(\mathcal{P}(R),\subseteq)$, ayant pour propriété de conserver l'ordre des éléments, i.e.

$$X \preceq Y \Longleftrightarrow f(X) \subseteq f(Y)$$

Si de plus la fonction f^{-1} existe et est calculable, on dit alors que $(\mathcal{L},\preceq)$ (ou simplement $\mathcal{L}$ quand $\preceq$ est implicite dans le contexte) *admet une représentation ensembliste*. Notons que l'espace de recherche des motifs intéressants peut alors être représenté par le treillis des parties de R, i.e. $\mathcal{P}(R)$.

Cette représentation ensembliste permet de relier ce type de problème aux transversaux minimaux d'un hypergraphe. Pour rappel, soit V un ensemble fini d'éléments. Un *hypergraphe* sur V est une famille $\mathcal{H} = \{E_1, E_2, ...E_m\}$ de parties de V telle que $E_i \neq \emptyset$ $(i = 1, 2, ..m)$ et $\bigcup_{i=1}^{m} E_i = V$. Un *hypergraphe simple* est tel qu'aucune arête n'est contenue dans une autre arête. On appelle *hypergraphe simple associé* à $\mathcal{H}$ l'hypergraphe $\mathcal{H}'$ composé des éléments minimaux par inclusion de $\mathcal{H}$. Un *transversal* T de $\mathcal{H}$ est un sous-ensemble de V qui intercepte tous les éléments de $\mathcal{H}$. T est *minimal* s'il ne contient aucun autre transversal de $\mathcal{H}$. L'ensemble de tous les transversaux minimaux de $\mathcal{H}$ est noté $Tr(\mathcal{H})$. Si $\mathcal{H}'$ est l'hypergraphe simple associé à $\mathcal{H}$, alors $Tr(\mathcal{H}) = Tr(\mathcal{H}')$.

La complexité de ce problème reste encore inconnue. Un algorithme sub-exponentiel pour résoudre ce problème a été proposé dans [FK96], et plusieurs cas particuliers peuvent être résolus en temps polynomial [EG95].

Supposons que (f, R) représente $\mathcal{L}$ sous sa forme ensembliste. Soit $S = Th(\mathcal{L}, r, Q)$. On a alors le théorème suivant :

Théorème 1: [MT97]

$$\mathcal{B}d^-(S) = f^{-1}(\ Tr(\overline{f(\mathcal{B}d^+(S))})\ ),\ avec\ \overline{f(\mathcal{B}d^+(S))} = \{R \backslash f(\varphi)\ /\ \varphi \in \mathcal{B}d^+(S)\}$$

De façon surprenante, les auteurs de [MT97] n'ont pas, à notre connaissance, exploité la propriété duale dans aucune de leur publication, i.e. celle qui exprime la bordure positive en fonction de la bordure négative. Cette propriété duale a été proposée dans [MP03] pour la première fois. Nous redonnons ici ce résultat mais avec une preuve différente.

Théorème 2:

$$\begin{aligned} \mathcal{B}d^+(S) = f^{-1}(\ & \overline{Tr(f(\mathcal{B}d^-(S)))}\ ) \\ avec\ & \overline{Tr(f(\mathcal{B}d^-(S)))} \ = \{R \backslash u\ /\ u \in Tr(f(\mathcal{B}d^-(S)))\} \end{aligned}$$

Preuve
Soient $S = Th(\mathcal{L}, r, Q)$ et (f, R) la représentation ensembliste de $\mathcal{L}$. D'après [MT97], on a : $\mathcal{B}d^-(S) = f^{-1}(\ Tr(\overline{f(\mathcal{B}d^+(S))})\ )$, avec $\overline{f(\mathcal{B}d^+(S))} = \{R \backslash f(\varphi)\ /\ \varphi \in \mathcal{B}d^+(S)\}$.

De plus, on a $Tr(Tr(\mathcal{H})) = \mathcal{H}$, avec $\mathcal{H}$ un hypergraphe simple [Ber73].

Or

$$\begin{aligned} Tr(f(\mathcal{B}d^-(S))) &= Tr(Tr(\overline{f(\mathcal{B}d^+(S))})) \\ &= \overline{f(\mathcal{B}d^+(S))} \end{aligned}$$

On obtient donc :

$$\begin{aligned} \mathcal{B}d^+(S) = f^{-1}(\ & \overline{Tr(f(\mathcal{B}d^-(S)))}\ ) \\ avec\ & \overline{Tr(f(\mathcal{B}d^-(S)))} \ = \{R \backslash u\ /\ u \in Tr(f(\mathcal{B}d^-(S)))\} \end{aligned}$$

Un grand nombre et une grande variété de problèmes rentrent dans ce cadre [MT97] : des problèmes d'extraction de motifs fréquents ou de leur représentations condensées, à la découverte de contraintes d'intégrités dans des bases de données relationnelles, en passant par l'étude de fonctions booléennes ou des phases d'un processus de réécriture de requêtes [JPR+05]. Certaines de ces applications seront plus amplement présentées en section 8.1.

Chapitre 4

Les principales stratégies de recherche

Sommaire

L'extraction des motifs étant une tâche complexe, la stratégie employée pour parcourir cet espace a donc une très grande importance. Nous allons plus particulièrement nous focaliser sur les stratégies employées pour l'extraction des motifs fréquents, ce problème étant le plus étudié. Toutefois, ces stratégies sont dans l'ensemble génériques et peuvent être appliquées à tout problème reformulable dans le cadre précédemment introduit.

Les algorithmes proposés pour résoudre ce problème sont construits autour de deux principales stratégies d'exploration de l'espace de recherche : ceux effectuant un parcours par niveau [AS94, Bay98, LK98] et ceux effectuant un parcours en profondeur de l'espace de recherche [ZPOL97, HPY00, BCG01, GZ01, UAUA03].

Dans ce chapitre, nous allons étudier ces différentes stratégies ainsi que leurs principales variantes. En général, les algorithmes de découverte de motifs fréquents utilisent aussi des propriétés spécifiques au support pour élaguer l'espace de recherche et/ou éviter des accès aux données. Notre objectif dans ce chapitre est de rester général et de nous intéresser, indépendamment du prédicat, uniquement au parcours effectué par les algorithmes, i.e. à l'ensemble de motifs testés par rapport au prédicat étudié. Les techniques propres au support ne seront donc pas détaillées ici.

4.1 Parcours par niveau

4.1.1 Parcours par niveau classique

Dans le reste de cette thèse, nous supposerons pour simplifier que les articles dans les transactions et dans les motifs sont ordonnés lexicographiquement. De plus, un motif {A,B,C,D} sera noté ABCD.

Un des algorithmes type effectuant un parcours par niveau de l'espace de recherche est *Apriori* [AS94]. Plus précisément, cet algorithme effectue un parcours en largeur des motifs les plus petits de l'espace de recherche aux plus grands, et utilise la propriété d'anti-monotonie du prédicat pour élaguer l'espace de recherche, i.e. $X \notin F \Rightarrow \forall Y \supset X, Y \notin F$.

L'algorithme commence par rechercher les articles fréquents. Puis, pour chaque itération k, il génère un ensemble de motifs *candidats* de taille k, noté C_k. Ces motifs candidats sont des motifs potentiellement fréquents. Pour élaguer l'espace de recherche et limiter le nombre de motifs candidats générés, les motifs candidats se limitent à ceux pour lesquels tous leurs sous-ensembles de taille $k-1$ sont fréquents. Une fois cet élaguage effectué, l'algorithme teste chacun des motifs candidats, et utilise les k-motifs fréquents découverts pour débuter l'itération suivante. L'algorithme s'arrête lorsque l'ensemble des motifs candidats est vide.

L'exemple suivant illustre le fonctionnement de cet algorithme.

Exemple 3:
Soit la base de données de transactions r suivante : {ABCDF, BDEF, BEF, BDF, BDE, EF, C, CF, EF, DE, AC, CD }

La figure 4.1 représente le parcours de l'espace de recherche effectué par un algorithme par niveau tel qu'Apriori pour découvrir l'ensemble des motifs fréquents pour un seuil minimum de support de 1. Les motifs en noir sont les motifs fréquents générés, et en gris les motifs de l'espace de recherche non générés. Les motifs AE et CE (hachurés sur la figure) sont les deux seuls motifs non fréquents générés.

L'algorithme commence par rechercher les motifs fréquents de taille 1, i.e. F_1 ={A, B, C, D, E, F}. Il utilise ensuite ces motifs pour générer les motifs candidats de taille 2. Tous les motifs candidats de taille 2 sont fréquents à l'exception de AE et CE. La troisième itération utilise les motifs fréquents de taille 2 pour générer des motifs candidats de taille 3, en supprimant des motifs candidats tous ceux ayant pour sous-ensemble AE et CE. Les motifs candidats sont ensuite testés, et les fréquents sauvegardés. L'algorithme continue jusqu'à la découverte de ABCDF. Il effectue donc au total 4 itérations, chacune explorant un niveau de l'espace de recherche.

Cet algorithme a notamment été utilisé pour étudier l'impact de techniques d'implémentations et de certaines heuristiques sur les performances [Bod03, Bor03, Bod04]. Ces

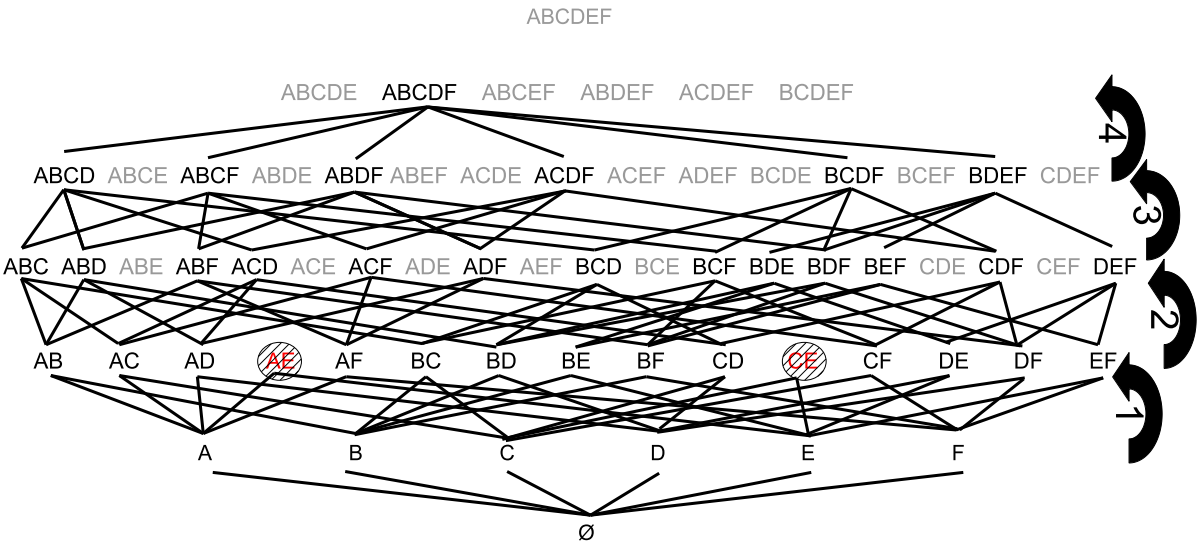

FIG. 4.1 – Exemple d'approche par niveau (*Apriori*)

travaux ont notamment montré qu'ordonner les articles par support croissant permettait d'améliorer fortement les performances. Ils ont aussi souligné l'intérêt de structures de données de type arbres préfixés (ou *trie*) pour la génération des motifs candidats. De manière générale, ces travaux ont permis de mettre en avant certains principes pouvant bénificier à l'ensemble des algorithmes d'extraction de motifs.

4.1.2 Parcours par niveau avec "sauts" dans l'espace de recherche

D'autres algorithmes utilisent un parcours par niveau du même type qu'*Apriori* tel que $MaxMiner$ [Bay98]. A la différence d'*Apriori* qui effectue uniquement un parcours par niveau de l'espace de recherche, ces algorithmes font en plus des "sauts" dans l'espace de recherche. Ces "sauts" consistent à générer et tester des motifs de taille supérieure à k, où k est le niveau en cours d'étude. Grâce à la propriété d'anti-monotonie du prédicat, les motifs fréquents découverts lors de ces "sauts" sont ensuite utilisés pour élaguer l'espace de recherche. $MaxMiner$ parcourt les motifs par niveau en exploitant le système d'énumération de Rymon [Rym92], comme représenté dans la figure 4.2 pour l'exemple précédent. La principale caractéristique de cet algorithme est de calculer, en même temps que le support d'un motif M (par exemple BD sur la figure 4.2), le support du plus grand motif apparaissant dans le sous-arbre de M (soit $BDEF$ dans la figure 4.2). Si celui-ci est fréquent, alors tout le sous-arbre dont la racine est M est élagué pour la visite des prochains niveaux.

L'algorithme $Pincer - Search$ [LK98] effectue aussi un parcours par niveau avec des sauts. Mais à la différence de $MaxMiner$, les sauts effectués ne sont pas dépendants de l'ordre des éléments. Pour chaque niveau exploré, $Pincer - search$ génère les plus

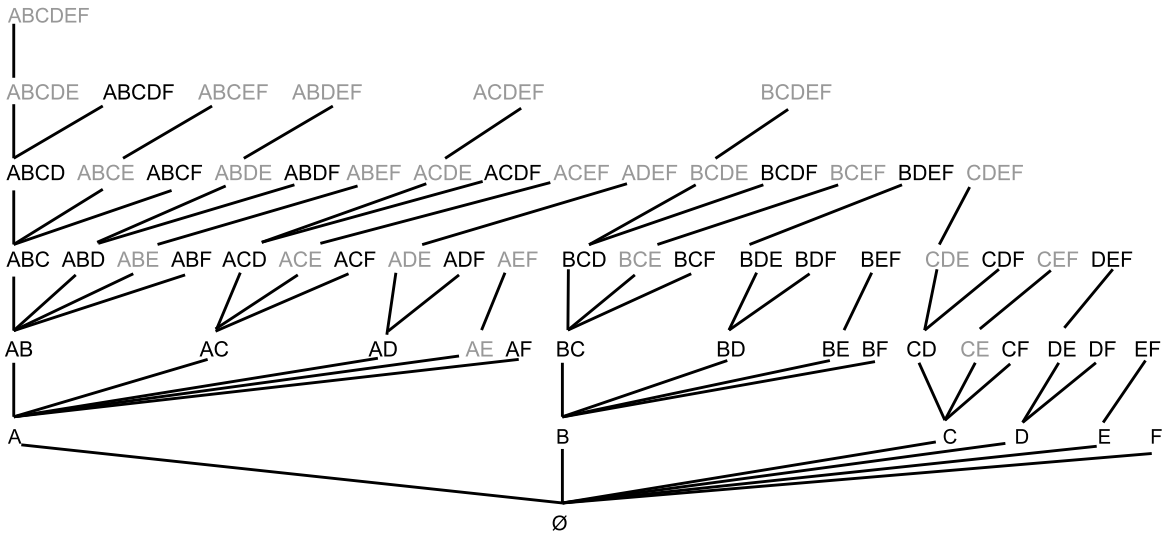

FIG. 4.2 – Système d'énumération de Rymon

grands motifs n'incluant pas un motif non fréquent déjà découvert lors des itérations précédentes. Intuitivement, ces motifs sont donc les plus grands motifs potentiellement fréquents à une itération donnée. Les motifs fréquents maximaux découverts sont ensuite utilisés pour élaguer l'espace de recherche. Après avoir fait un saut à partir des motifs découverts au niveau k, $Pincer - Search$ supprime les motifs fréquents de taille k inclus dans un motif fréquent maximal découvert. Or, la méthode de génération des candidats de $Pincer - search$ est celle utilisée par *Apriori*, i.e $Y \in C_{k+1}$ si et seulement si $\forall X \in C_k, X \subseteq Y, X \in F$. Par conséquent, l'élagage fait à partir des sauts peut supprimer un motif de C_k nécessaire par la suite à la génération d'un motif de $\mathcal{B}d^+(F)$ non encore découvert. Par exemple, supposons que $R = \{I, J, K, L, M\}$ et que $IJK \in \mathcal{B}d^+(F)$ est découvert dès le deuxième niveau. L'algorithme va donc supprimer tout ses sous-ensembles de taille 2, dont JK. Par conséquent, l'algorithme ne pourra pas générer JKL et tout ces sur-ensembles, alors que $JKLM$ peut très bien être un motif de $\mathcal{B}d^+(F)$. Face à ce problème, $Pincer - Search$ doit donc regénérer certains motifs de taille k pour pouvoir découvrir l'ensemble des motifs de la bordure positive.

4.2 Parcours en profondeur

Un grand nombre d'algorithmes [ZPOL97, HPY00, BCG01, GZ01, UAUA03] effectuent un parcours en profondeur de l'espace de recherche. Il existe plusieurs types de parcours en profondeur.

4.2.1 Parcours en profondeur classiques

Dans ce type de parcours, l'espace de recherche est exploré suivant un parcours en profondeur d'après le système d'énumération de Rymon. Le premier algorithme à utiliser une stratégie de parcours en profondeur est l'algorithme *Eclat* [ZPOL97, Zak00]. Notons qu'en général le parcours effectué n'est pas un "pur" parcours en profondeur. En effet, les algorithmes explorent à chaque itération tous les motifs ayant le même préfixe, afin de les réordonner en fonction du support. Cette technique permet de supprimer des motifs non fréquents ainsi que leurs sur-ensembles et de trouver plus rapidement les grands motifs fréquents.

La figure 4.3 montre une partie du parcours en profondeur effectué pour l'exemple 3. Les motifs fréquents générés apparaissent en noir, les motifs non fréquents générés en hachuré, et les motifs de l'espace de recherche non générés en gris. Les ensembles de motifs entourés correspondent aux premières itérations de l'algorithme. Pour faciliter la lisibilité du schéma, seules les premières itérations sont prises en compte.

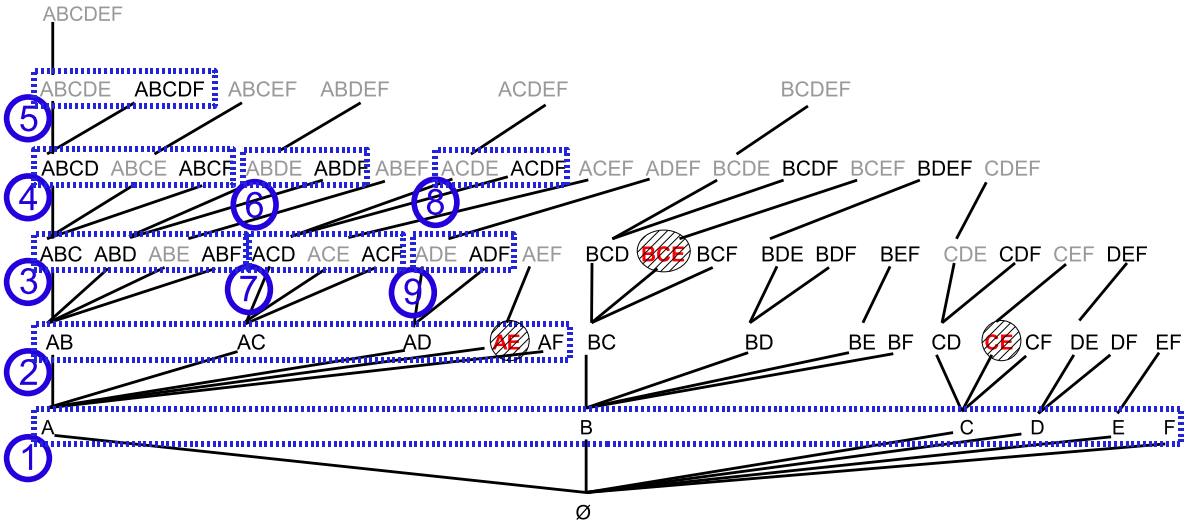

FIG. 4.3 – Exemple de parcours en profondeur

Par rapport à une approche par niveau de type *Apriori*, une approche en profondeur génère et teste plus de motifs candidats. En effet, lors du parcours en profondeur, le motif non fréquent BCE a été généré, alors que le motif CE est non fréquent. Toutefois, les approches en profondeur ont deux principaux avantages par rapport aux approches par niveau :

- elles traitent à chaque itération moins de motifs en mémoire (le nombre de motifs en mémoire est polynomial et non exponentiel comme pour les approches par niveau) ;
- elles trouvent plus rapidement de grands motifs fréquents.

Afin d'exploiter ces avantages, des variantes de cette approche ont été développées telles que *Mafia* [BCG01] ou *GenMax* [GZ01]. En plus d'optimiser le comptage du sup-

port, ces algorithmes utilisent les motifs fréquents maximaux pour élaguer l'espace de recherche. $Mafia$ utilise de plus une propriété du support pour tester moins de motifs. Cette propriété, appelée *Parent Equivalent Pruning*, permet à l'algorithme de ne pas tester tous les motifs fréquents d'une classe d'équivalence. Plus précisemment, l'alghorithme va uniquement générer et tester les motifs X tel que $prefix_i(Cl(X)) = X$, $\forall i \in \{1, .., |Cl(X)|\}$, où $prefix_i(Z)$ correspond aux i premiers motifs de Z. Même si les techniques employées sont différentes, dans l'ensemble la stratégie est la même : supprimer tout sous-arbre dont le plus grand motif est inclus dans un motif fréquent maximal déjà découvert.

4.2.2 Cas particulier des motifs fréquents

Parcours en profondeur à partir des transactions

L'algorithme $FP - Growth$ [HPY00] et ses variantes font aussi un parcours en profondeur de l'espace de recherche. Mais à la différence des algorithmes précédents, ce type d'algorithmes n'effectue pas de phase de génération des motifs candidats, suivie d'une phase de vérification de la fréquence. Ils génèrent directement les motifs et leur support à partir des transactions, ce qui restreint l'approche à ces problèmes.

Pour y parvenir, l'algorithme s'appuie sur une structure de données appelée $FP-tree$ (*Frequent Pattern Tree*). Cette structure est composée d'un arbre préfixé (ou *trie*), et d'une liste transversale de pointeur faisant le lien entre les articles et les transactions. Les articles contenus dans la liste et dans l'arbre sont les articles fréquents ordonnés par support croissant, ce qui permet d'avoir une représentation des données compacte en mémoire. La figure 4.4 montre le $FP-tree$ créé au début de l'algorithme pour l'exemple 3.

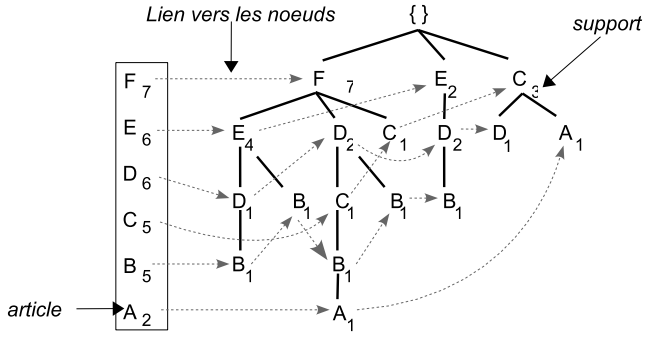

FIG. 4.4 – Exemple de $FP - tree$

L'algorithme utilise cette structure pour effectuer de manière récursive des projections sur la base de données et effectuer un parcours en profondeur des motifs apparaissant dans les transactions. Chaque itération correspond alors à la construction d'un $FP - tree$. La figure 4.5 montre le parcours de l'espace de recherche effectué par l'algorithme $FP - growth$ sur l'exemple 3. Les motifs en noir sont les motifs fréquents générés. Les

motifs en gris sont des motifs non fréquents qui n'ont pas été générés par l'algorithme, mais qui l'avaient été par un parcours en profondeur classique (voir figure 4.3). Les motifs générés à chaque itération de l'algorithme apparaissent dans un cadre en pointillé sur la figure avec le numéro de l'itération en dessous.

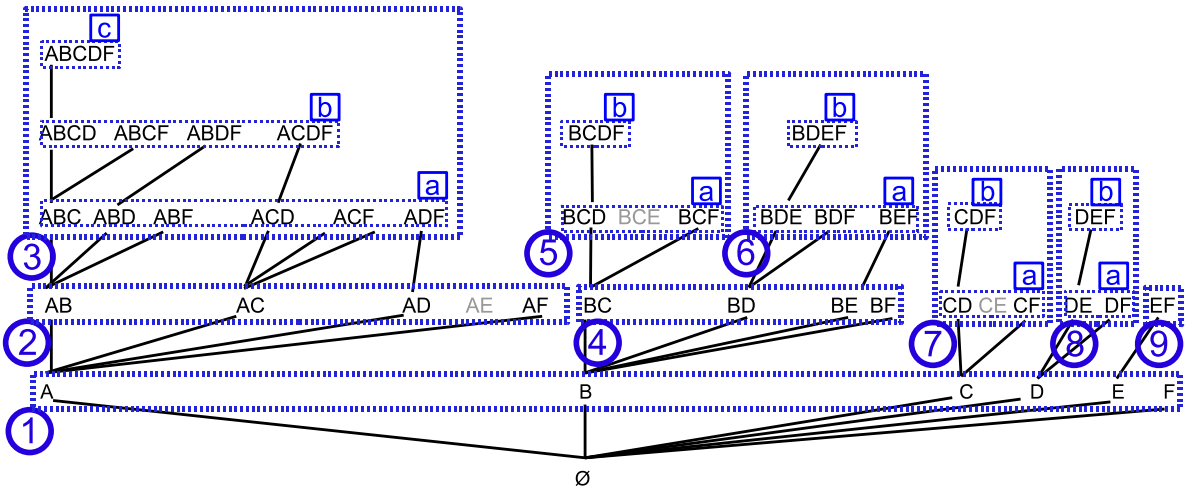

FIG. 4.5 – Parcours de l'espace de recherche effectué par $FP-Growth$

Comme le montre la figure 4.5, l'algorithme trouve l'ensemble des motifs fréquents en neuf itérations. Le parcours effectué correspond dans l'ensemble à celui effectué lors d'un parcours en profondeur classique. Toutefois, certaines itérations permettent de générer directement tout un sous-arbre, c'est le cas dans notre exemple des itérations 3, 5, 6, 7 et 8. Elles correspondent à la création d'un $FP-tree$ composé uniquement d'un seul chemin. Dans ce cas, l'algorithme arrête sa récursion et génère directement les motifs constitués du motif étudié et des articles du $FP-tree$ car ils sont par construction nécessairement fréquents. La figure 4.6 représente ce type d'itération pour l'exemple 3, lors de l'étude du sous-arbre de l'article A.

De plus, le parcours effectué par ce type d'algorithme évite de générer des motifs n'apparaissant pas dans les transactions. Par exemple, les motifs AE, BCE et CE ne sont jamais explorés.

Des variantes de cet algorithme ont été développées notamment pour découvrir les motifs fréquents maximaux. L'algorithme $FPmax*$ est un de ces algorithmes [GZ03]. Lorsqu'un $FP-tree$ est composé d'un seul chemin, cet algorithme génère directement un motif fréquent maximal en faisant tout simplement l'union du motif en cours d'étude avec l'ensemble des articles du $FP-tree$. Il effectue ainsi un "saut" dans l'espace de recherche et évite de générer des motifs, comme le montre la figure 4.6 où $ABCDF$ est généré à partir de A et des articles de $FP-tree_A$. Ces motifs fréquents maximaux sont stockés et ensuite utilisés pour élaguer l'espace de recherche.

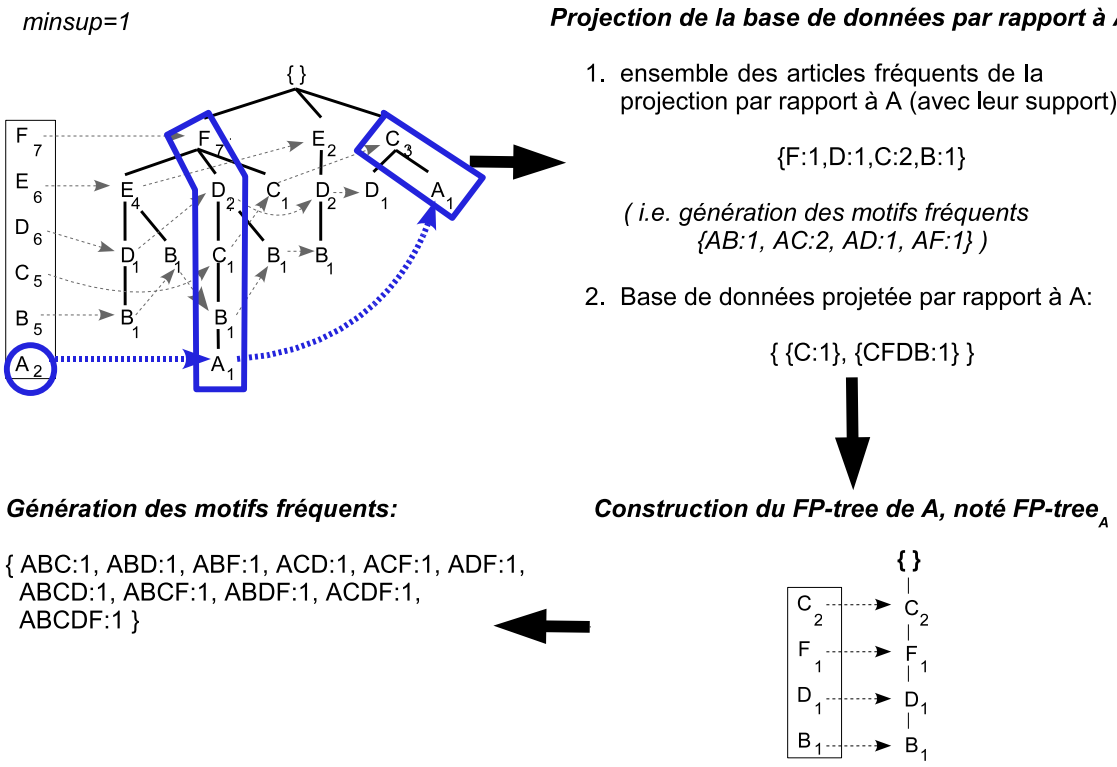

FIG. 4.6 – Exemple d'itération de $FP-growth$

Parcours en profondeur à partir des motifs fermés

Des algorithmes tels que LCM [UAUA03] n'effectuent pas un parcours de l'espace de recherche à partir des motifs fréquents. Ils exploitent à la place les motifs fermés.

LCM s'appuie sur la notion d'extension de fermeture (*closure extension*) d'un motif pour parcourir l'espace de recherche d'un motif fermé à l'autre, évitant ainsi de tester un grand nombre de candidats. Un motif Q est une extension de fermeture d'un motif P si $Q = Cl(P \cup \{i\}), i \notin P$. Comme les auteurs l'ont démontré [UAUA04], tout motif fermé ($\neq \emptyset$) est l'extension de fermeture d'un ou de plusieurs motifs fermés. Par conséquent, l'ensemble de ces motifs peut être énuméré itérativement en utilisant uniquement les fermés découverts lors de l'itération précédente.

Toutefois, à partir de cette propriété, un motif fermé peut être généré plusieurs fois. Afin d'éviter cela, la notion d'extension de fermeture conservant le préfixe (*prefix-preserving closure extension*), noté $ppc-extension$, a été introduite. Pour simplifier les notations, supposons que l'ensemble des articles est $R = \{1, 2, ..., n\}$. Soient P un motif et i un article de R . Notons $prefix_i(P)$ le sous-ensemble de P composé seulement des articles plus petits ou égaux à i dans R , i.e. $prefix_i(P) = P \cap \{1, 2, ..., i\}$. La $ppc-extension$ d'un motif fermé P peut alors être défini de la manière suivante :

Définition 9:
Un motif fermé Q est une $ppc-extension$ d'un motif fermé P si :
- $Q = Cl(P \cup \{i\}), i \notin P$,
- $i > core\_i(P)$, où $core\_i(P)$ est le plus petit article de P tel que $Cl(P(i)) = P$,
- $prefix_{i-1}(P) = prefix_{i-1}(Q)$, i.e. le préfixe de P est préservé.

Théorème 3: [UAUA04]
Soit un motif fermé $Q \neq \emptyset$. Alors, il existe un ***unique*** *motif fermé P tel que Q est une $ppc-extension$ de P.*

Ainsi l'ensemble des motifs fermés peut être représenté sous forme d'arbre, où chaque noeud fils est la $ppc-extension$ de son noeud père. L'algorithme LCM développé parcourt cet arbre en profondeur en élaguant les sous-arbres issus d'un motif fermé non fréquent, et permet ainsi d'obtenir l'ensemble des motifs fermés fréquents.

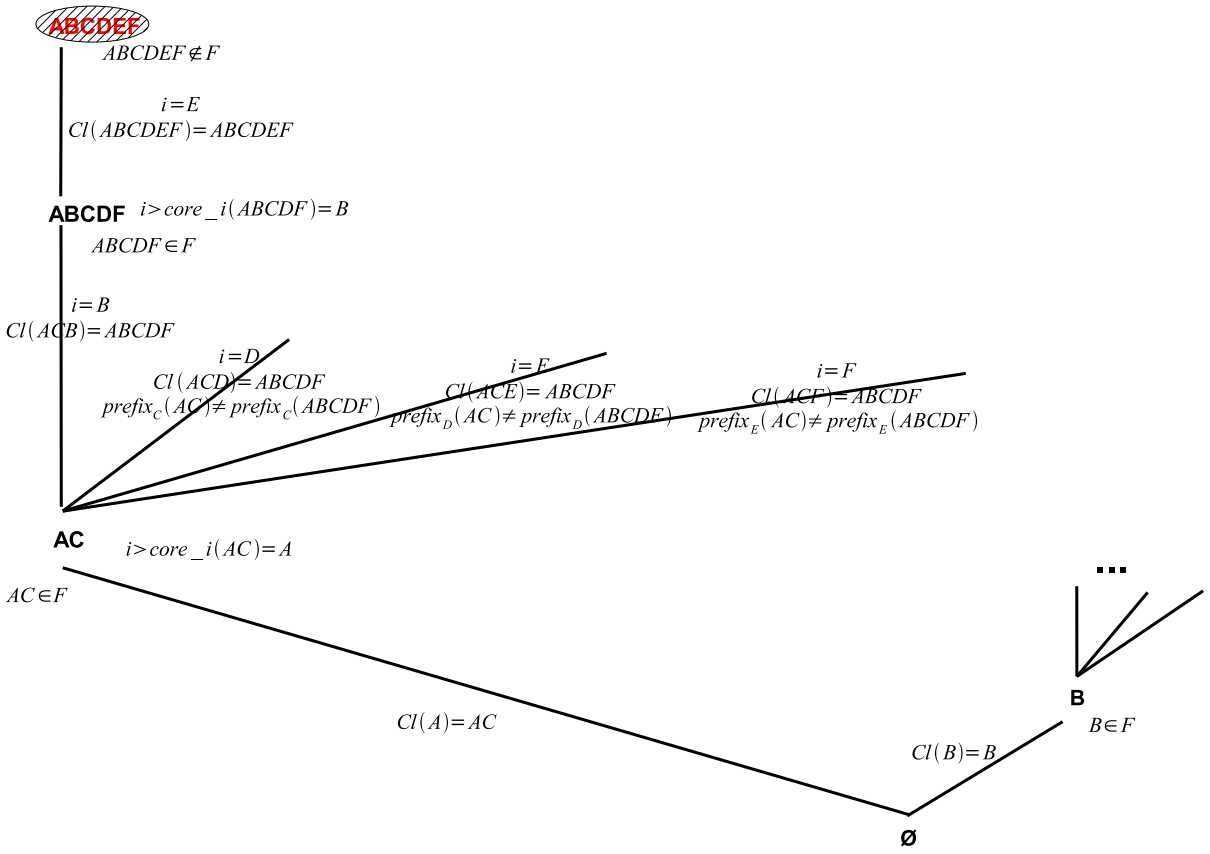

FIG. 4.7 – Exemple partiel d'exécution de LCM

La figure 4.7 montre les motifs générés lors des premières itérations de l'algorithme à partir des données de l'exemple 3. L'algorithme est initialisé avec les fermés de chaque article. Le premier motif généré est donc AC (le fermé de A). Ce motif étant fréquent, l'algorithme calcule ensuite sa $ppc-extension$ issue de $AC \cup \{B\}$, i.e. $ABCDF$. Ce

motif est fréquent, sa $ppc - extension$ $ABCDEF$ est donc construite. $ABCDEF$ est non fréquent. L'algorithme arrête donc l'exploration de cette sous-branche, et génère la $ppc - extension$ de $ABCDF$ suivante. Aucune autre $ppc - extension$ ne peut être créée pour ce motif. L'algorithme étudie donc les autres $ppc - extension$ du motif ayant permis de le générer, i.e. AC. Toutefois, les fermés obtenus à partir des articles D, E et F ne sont pas des $ppc - extension$ de AC car ils ne respectent pas la condition sur le préfixe. Par conséquent, l'algorithme a fini l'étude de la sous-branche de A, et continue son exploration du reste de l'arbre. Le parcours effectué au final par cet algorithme est illustré dans la figure 4.8 (les motifs hachurés sont les motifs non fréquents générés).

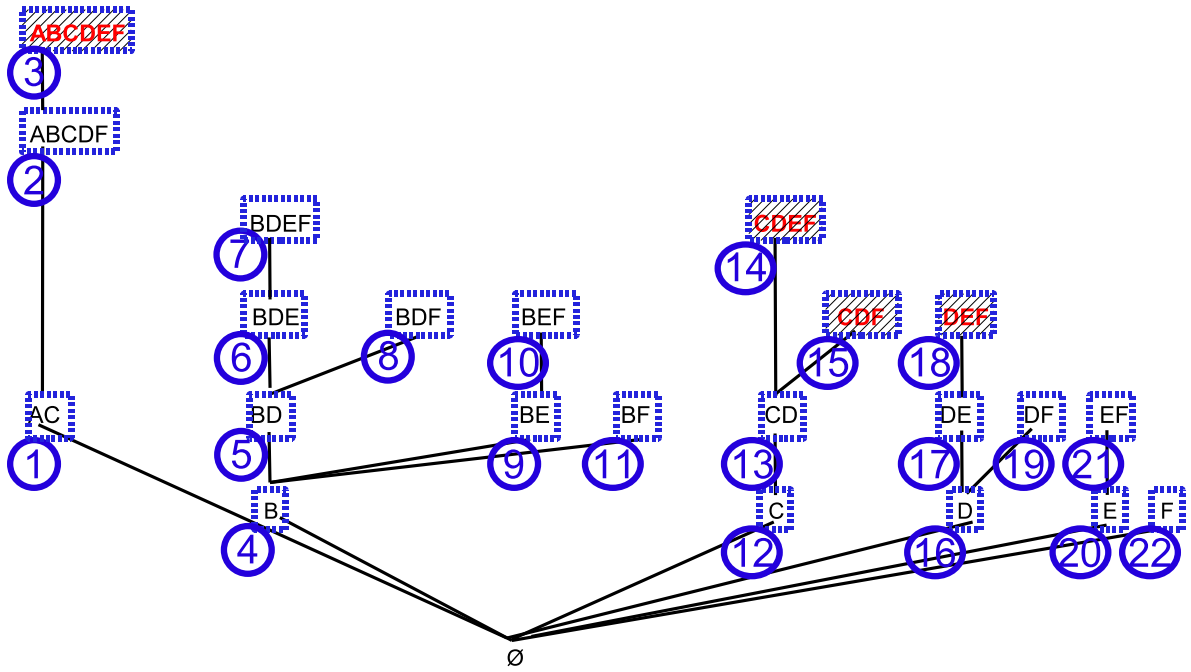

FIG. 4.8 – Parcours de l'espace de recherche effectué par LCM

A partir des motifs fermés fréquents découverts, il est ensuite possible de générer l'ensemble des motifs fréquents ainsi que leur support. Par ailleurs, certaines variantes de cet algorithme permettent de générer efficacement ces motifs et leur support.

D'autres versions de cet algorithme [UKA04] exploitent les motifs fréquents maximaux trouvés par le parcours en profondeur pour élaguer l'espace de recherche. En effet, rappelons que les motifs fréquents maximaux sont les plus grands motifs fréquents fermés, ils sont donc découverts lors de ce parcours.

4.3 Parcours s'appuyant sur la notion de dualisation

L'algorithme *Dualize and Advance* [GKM+03] exploite la notion de *dualisation* pour effectuer des sauts dans l'espace de recherche.

4.3 Parcours s'appuyant sur la notion de dualisation

Cette notion s'appuie sur les théorèmes 1 et 2 présentés en section 3, permettant de relier les bordures par un calcul de transversaux minimaux. Une dualisation consiste alors à utiliser ces théorèmes pour passer d'une bordure à l'autre.

Dans leur algorithme, les auteurs utilisent plus particulièrement le théorème 1 pour estimer la bordure négative à partir d'une bordure positive en construction, qui est toujours un sous-ensemble de la bordure positive à découvrir. À chaque itération, la bordure négative qui correspond à ce sous-ensemble est générée par un calcul de transversaux minimaux. Les éléments de cette bordure négative sont testés sur les données jusqu'à en trouver un qui soit intéressant ; celui-ci est alors le point de départ d'une recherche aléatoire en profondeur jusqu'à la découverte du motif de la bordure positive qui le spécialise. Une fois trouvé, ce nouveau motif est ajouté à la bordure positive en construction qui est utilisée pour une nouvelle itération. L'algorithme s'arrête lorsque la bordure négative est entièrement découverte.

Notons que la bordure négative générée par la dualisation est constituée des plus petits motifs non inclus dans un motif de la bordure positive en construction.

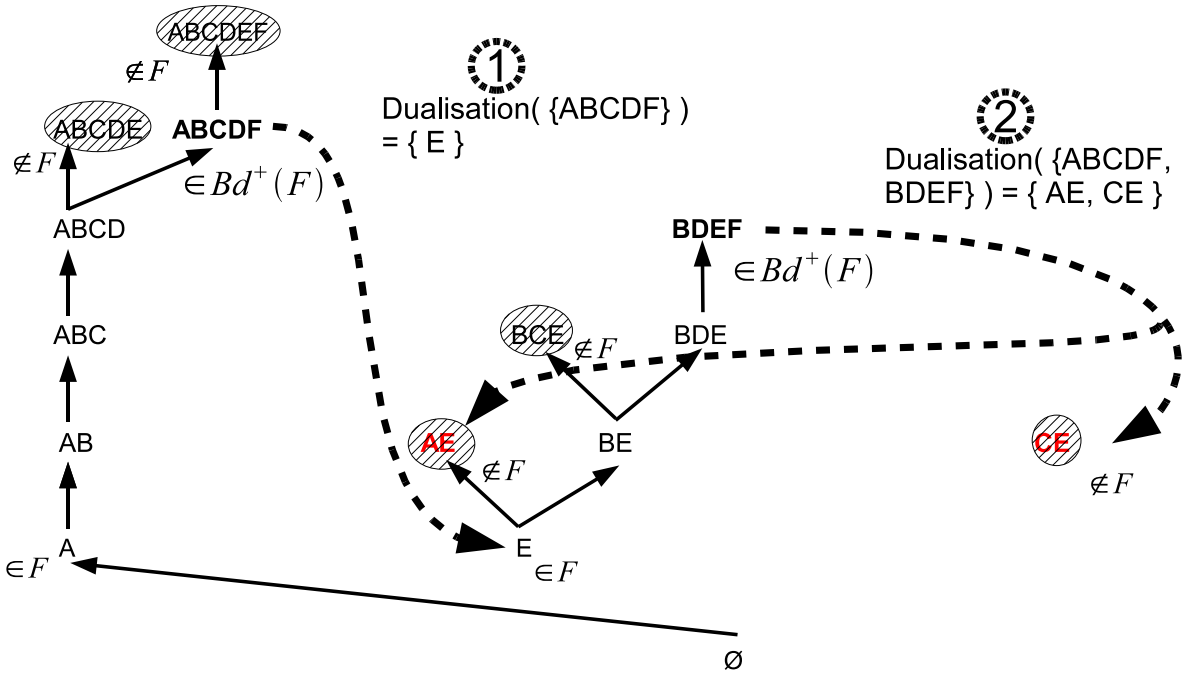

FIG. 4.9 – Parcours de l'espace de recherche effectué par *Dualize and Advance*

La figure 4.9 illustre l'exécution de l'algorithme à partir des données de l'exemple 3. La première étape de l'algorithme consiste à effectuer un parcours en profondeur à partir du premier article fréquent testé, i.e. A, jusqu'à la découverte d'un motif fréquent maximal. A l'issue de ce parcours en profondeur, le motif $ABCDF \in \mathcal{B}d^+(F)$ est découvert et est ajouté à la bordure positive en constuction. A partir de cette bordure, la première dualisation est faite et génère le motif E. Ce motif est fréquent, l'algorithme fait donc un parcours en profondeur jusqu'à la découverte du motif fréquent maximal $BDEF$. La

deuxième dualisation se fait donc à partir de l'ensemble $\{ABCDF, BDEF\}$. Elle génère les motifs AC et AE qui sont tous deux non fréquents. Par conséquent, l'algorithme s'arrête et la bordure positive finale est $\{ABCDF, BDEF\}$.

Notons que cet algorithme fait une dualisation par motif fréquent maximal.

Le tableau 4.1 fait une synthèse des différentes stratégies présentées dans ce chapitre. Pour chaque algorithme, ce tableau présente le type de parcours effectué, la (les) solution(s), et les stratégies d'élagage et de sauts utilisées. Les solutions présentées ici sont celles découvertes par les algorithmes sans aucun traitement supplémentaire. Des variantes de ces algorithmes ont aussi été développées afin de trouver efficacement d'autres solutions.

Algorithme	**Type de parcours**	**Solution(s)**	**Stratégie d'élagage** à chaque itération	**Stratégie de sauts** à chaque itération
Apriori	largeur	la théorie et la bordure négative	élagage à partir des motifs de la bordure négative	
MaxMiner	largeur	les bordures positive et négative	élagage à partir des motifs de la bordure négative	test le plus grand motif apparaissant dans le sous-arbre du motif en cours d'étude
Pincer-Search	largeur	les bordures positive et négative	élagage à partir des motifs de la bordure négative	test les plus grands motifs non invalidés par un motif de la bordure négative déjà découvert
Eclat	profondeur	la théorie	élagage à partir des motifs non fréquents découverts	
GenMax	profondeur	la bordure positive	élagage à partir des motifs non fréquents et des motifs de la bordure positive découverts	
Mafia	profondeur	la bordure positive	élagage à partir des motifs non fréquents, des motifs de la bordure positive, et des motifs fermés fréquents découverts	
FP-Growth	profondeur	l'ensemble des motifs fréquents	élagage à partir des motifs non fréquents découverts	
LCM	profondeur	l'ensemble des motifs fermés fréquents	élagage à partir des motifs non fermés fréquents découverts	sauts entre les générateurs fréquents et leurs fermés
Dualize and Advance	profondeur + dualisation	les bordures positive et négative	élagage à partir des motifs non fréquents découverts	saut de la bordure positive (en construction) à la bordure négative (en construction)

TAB. 4.1 – Comparatif des principaux algorithmes d'extraction de motifs

Dans la suite de ce manuscrit, nous allons aborder les problèmes suivants :

* L'adaptation des algorithmes d'extraction de motifs fréquents aux données. Face à ce problème, nous étudierons
  - le problème de la caractérisation des jeux de données, i.e. identifier les caractéristiques des jeux de données influençant les algorithmes ;
  - puis les limites des algorithmes adaptatifs existants. Nous allons plus particulièrement aborder le problème de la découverte adaptative des motifs fréquents maximaux.
* La mise en place de solutions génériques efficaces pour l'extraction de motifs intéressants. L'objectif ici est de tirer profit du grand nombre d'algorithmes proposés pour l'extraction de motifs fréquents pour résoudre toute une famille de problèmes de découverte de connaissances. Nous étudierons
  - dans un premier temps, l'application des algorithmes existants pour la recherche de motifs fréquents dans le cadre de la découverte de motifs intéressants ;
  - puis dans un second temps, le problème de l'utilisation des solutions logicielles existantes dans ce cadre générique.

Troisième partie

Algorithme adaptatif d'extraction de motifs fréquents

Chapitre 5

Contexte et état de l'art

Sommaire

Pour faire face à la grande variété de données et à l'influence de celles-ci sur les algorithmes, les algorithmes adaptatifs sont des solutions particulièrement prometteuses. Toutefois, leur mise en place implique une bonne maîtrise des stratégies existantes, ainsi que la connaissance des caractéristiques des données influençant ces stratégies.

Dans ce chapitre, nous allons présenter les principales caractéristiques étudiées des jeux de données et voir comment certaines sont utilisées dans les algorithmes pour mettre en place des stratégies adaptatives.

5.1 Caractéristiques des jeux de données

5.1.1 Caractéristiques des transactions

Les caractéristiques des transactions ont naturellement un fort impact sur les algorithmes. A ce niveau, les principales caractéristiques identifiées sont :

- le nombre, la taille, et le type des transactions ;
- le nombre d'articles.

De manière générale, plus la taille de la base de données est importante et ses transactions longues, plus le temps d'accès aux données pour déterminer le support des motifs aura tendance à être élevé. Une grande partie du temps d'exécution étant lié au calcul du support, le comportement des algorithmes sera donc directement influencé par ces critères. En plus du temps d'exécution, l'autre ressource critique est la mémoire. Or, l'espace mémoire occupé dépend aussi de ces facteurs. En effet, les algorithmes chargent une partie plus ou moins importante des données en mémoire. Malgré l'utilisation de structures de données optimisant cet espace occupé, celui-ci peut parfois être trop important et impliquer des accès disques très coûteux en mémoire virtuelle, voire même la saturer et entraîner l'arrêt de l'algorithme.

En dehors de ces considérations, la difficulté de l'extraction des motifs fréquents dépend en grande partie de la taille de l'espace de recherche à parcourir pour trouver la solution. Or d'un point de vue théorique, cette taille est bornée par 2^n où n est le nombre d'articles. L'observation de ce critère permet donc de connaître le nombre de motifs à traiter par l'algorithme dans le pire cas. Dans le même esprit, la taille de la plus grande transaction permet de borner celle des plus grands motifs fréquents, et donc d'avoir une idée de l'espace de recherche à parcourir. Par exemple, si une base de données est composée de petites transactions, les motifs fréquents correspondants seront petits, et l'espace de recherche à parcourir peu important.

Même si ces caractéristiques sont basiques, nous verrons dans la section 5.2 qu'un grand nombre d'algorithmes les utilisent avec succès pour déterminer des changements de stratégies en fonction des données. Toutefois, ces caractéristiques des transactions ne permettent pas de caractériser finement l'espace de recherche.

5.1.2 Notion de densité et classification des jeux de données

Dans le contexte des motifs fréquents, une définition de la densité d'un jeu de données a été proposée dans [GZ01]. Un jeu de données est dit *dense* s'il produit un grand nombre de longs motifs fréquents même pour des valeurs élevées de seuil minimum de support. Tous les sous-ensembles de ces motifs étant fréquents, la taille de l'espace de recherche des motifs fréquents sera importante. Les algorithmes parcourant une partie plus ou moins importante de cet espace de recherche (chapitre 4), leurs performances vont être influencées par ce critère.

Dans [GZ01], cette densité est représentée par la distribution des motifs fréquents maximaux, i.e. le nombre de motifs fréquents maximaux par niveau. Cette notion de densité a été étudiée expérimentalement sur des jeux de données réels et synthétiques issus de [Goeb]. A partir de cette notion de densité, les auteurs dans [GZ01] ont proposé une classification des jeux de données en quatre types :

- Le premier type de jeux de données (*Chess* et *Pumsb*, figure 5.1) est caractérisé par une distribution symétrique du nombre de motifs fréquents maximaux par rapport à leur taille. Comme le montre la figure 5.1, une grande partie des motifs fréquents

maximaux est de taille 10-11, le plus long n'excédant pas 24.

- Les jeux de données tels que *Connect* et *Pumsb∗* constituent le deuxième type de jeux de données (figure 5.2). Pour ces jeux, le nombre de motifs fréquents maximaux augmente progressivement jusqu'à un pic, puis décroît brutalement. Par rapport à la figure 5.1, la taille des motifs fréquents maximaux est plus longue, et peut atteindre 40. Néanmoins, la majeure partie des motifs fréquents maximaux est de taille 18-19.
- Le troisième type de jeux de données correspond aux jeux de données non denses (ou *éparses*). Ces jeux de données sont caractérisés par des motifs fréquents maximaux de petite taille. Leur nombre est donc très important pour les premiers niveaux puis chute très rapidement par la suite (figure 5.3).
- Le dernier type de jeux de données est un peu particulier (figure 5.4). Il correspond au jeu de données *Mushroom*. Le nombre de motifs fréquents maximaux reste faible jusqu'à un niveau élevé. Puis il y a une explosion du nombre de très grands motifs fréquents maximaux, cette explosion s'arrêtant brutalement peu après.

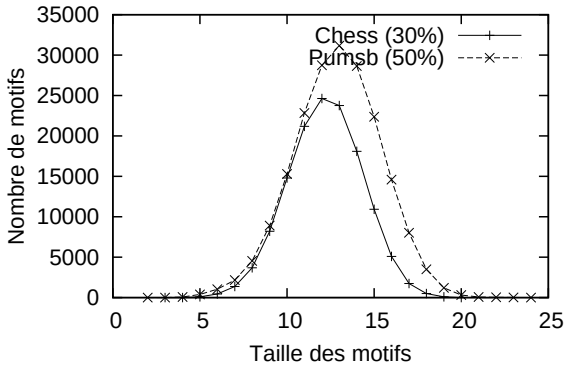

FIG. 5.1 – Jeux de données de type 1

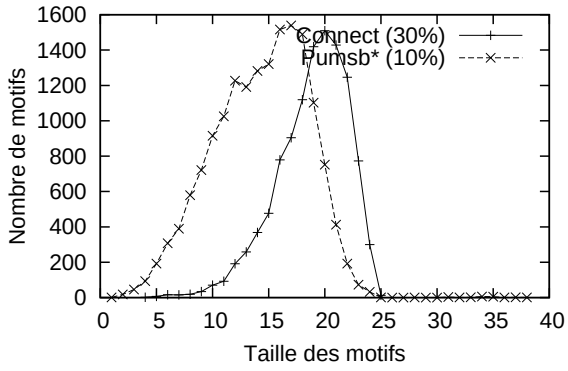

FIG. 5.2 – Jeux de données de type 2

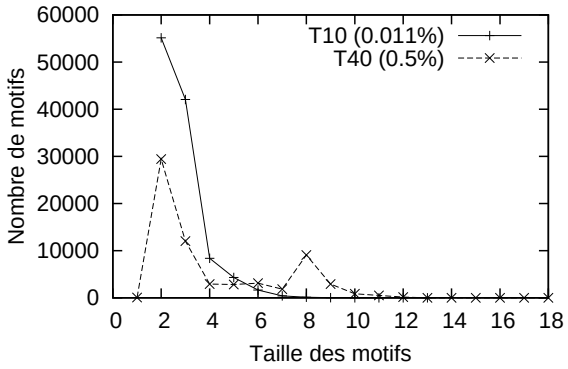

FIG. 5.3 – Jeux de données de type 3

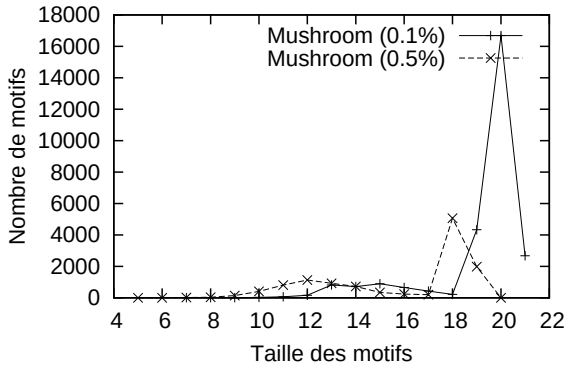

FIG. 5.4 – Jeux de données de type 4

La classification des jeux de données d'après la bordure positive des motifs fréquents est résumé dans le tableau 5.1.

Cependant, cette définition de la densité pose plusieurs problèmes.

Type	Type I	Type II	Type III	Type IV
Distribution de $\mathcal{B}d^+(F)$	symétrique	augmente progressivement puis décroît brutalement	explosion du nombre de petits motifs	explosion du nombre de très grands motifs
Taille des motifs de $\mathcal{B}d^+(F)$	grands	très grands	petits	très grands
Exemples de jeux de données	*Chess* (30%) *Pumsb* (50%)	*Connect* (30%) *Pumsb∗* (10%)	$T10I4D100K$ (0.011%)	*Mushroom* (0.1%)

TAB. 5.1 – Classification des jeux de données fondée sur la bordure positive des motifs fréquents

Limites de cette définition de la densité et de la classification

A la vue des figures 5.1 et 5.2, il apparaît que la différence entre les deux premiers types de jeux de données n'est pas évidente. Elle est principalement due à une légère différence de taille des motifs fréquents maximaux. Il est donc très difficile d'identifier le type de jeu de données.

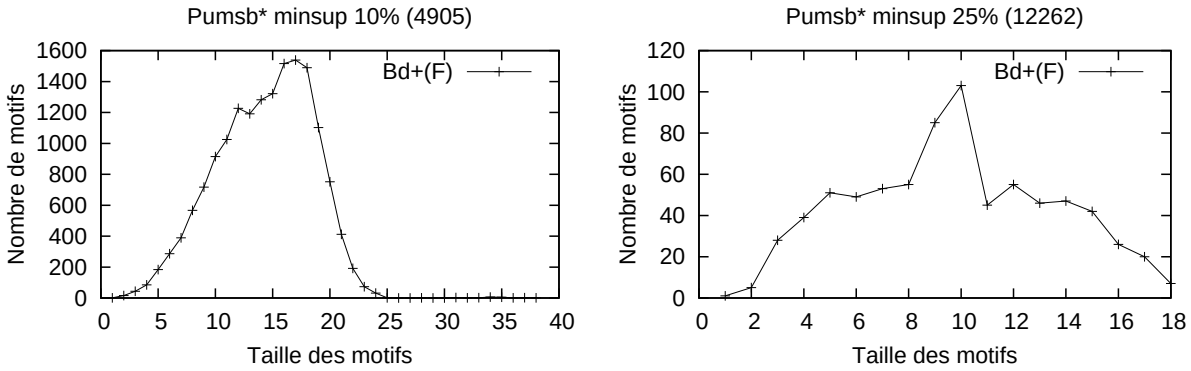

FIG. 5.5 – Bordure positive pour Pumsb* avec des minsup de 10% et de 25%

De plus, même si le type de jeu de données est identifié pour un seuil de support minimum donné, cette classification peut changer pour un autre seuil minimum de support. Etudions par exemple la densité, i.e. la distribution de la bordure positive des motifs fréquents, sur le jeu de données réels *Pumsb∗* pour deux seuils minimums de support (figure 5.5). Par exemple, pour un seuil minimum de support (relatif) de 9%, le nombre de motifs fréquents maximaux augmente progressivement jusqu'à un pic puis décroît brutalement. La taille des motifs fréquents maximaux est relativement élevée et peut atteindre 40. Avec la classification de [GZ01], ce jeu de données serait du deuxième type. Si on étudie ce même jeu de données pour un seuil de support différent, par exemple 25%, le plus grand motif fréquent est de taille 18 et la taille moyenne des motifs est de 10. La distribution de ce jeu est aussi changée, et devient relativement symétrique. Ainsi, *Pumsb∗* pourrait appartenir au premier type de [GZ01].

Cette remarque est aussi vraie pour le jeu de données *Mushroom* avec par exemple

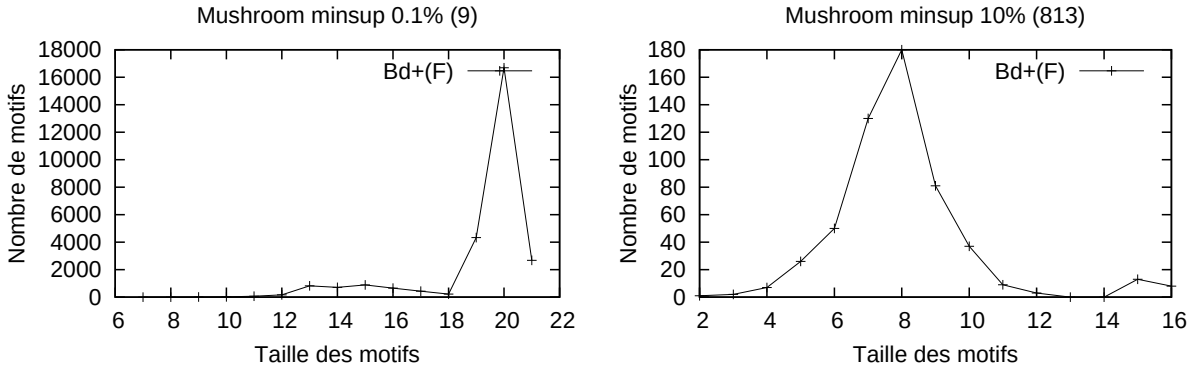

FIG. 5.6 – Bordure positive pour Mushroom avec des minsup de 0.1% et de 10%

des seuils de 0.1% et de 10% (figure 5.6). Pour le seuil de 0.1%, ce jeu de données est du type 4, alors qu'il est de type 1 pour un seuil de 10%. Il est donc nécessaire d'observer ces paramètres à plusieurs petits seuils minimums de support pour pouvoir mieux classer les jeux de données. Or pour ces valeurs de seuil, trouver la distribution de la bordure positive devient très difficile.

Des performances d'algorithmes en contradiction avec cette densité

Cette classification n'explique pas toujours les performances des algorithmes observées lors des bancs d'essais réalisés lors des ateliers FIMI [BZ03, BGZ04], i.e. les jeux de données les plus denses ne sont pas forcément les plus difficiles. Etudions par exemple les résultats obtenus pour certains algorithmes de découverte des motifs fréquents maximaux lors de $FIMI'04$ pour les jeux de données étudiés dans [GZ01] (figure 5.7).

Le tableau 5.2 représente approximativement le temps d'exécution moyen de ces algorithmes pour un seuil minimum de support fixé, ainsi que la classification de chaque jeu de données selon [GZ01].

Jeux de données	Seuil minimum de support	Type	Temps moyen d'exécution des algorithmes
Chess	30% (959)	type 1	$\approx$ 1 sec.
Connect	30% (20267)	type 2	$\approx$ 1 sec.
Pumsb	50% (24523)	type 1	1 à 10 sec.
Pumsb*	10% (4905)	type 2	1 à 10 sec.

TAB. 5.2 – Densité et performance des algorithmes

La classification de [GZ01] pourrait laisser penser que les jeux de données tels que *Connect* et *Pumsb*$*$ sont plus difficiles. En effet, la taille de leurs motifs fréquents maximaux est plus grande (figure 5.2). Leur densité est donc plus importante. Pourtant comme le montre le tableau 5.2, il semble ne pas y avoir de corrélation entre la classification issue

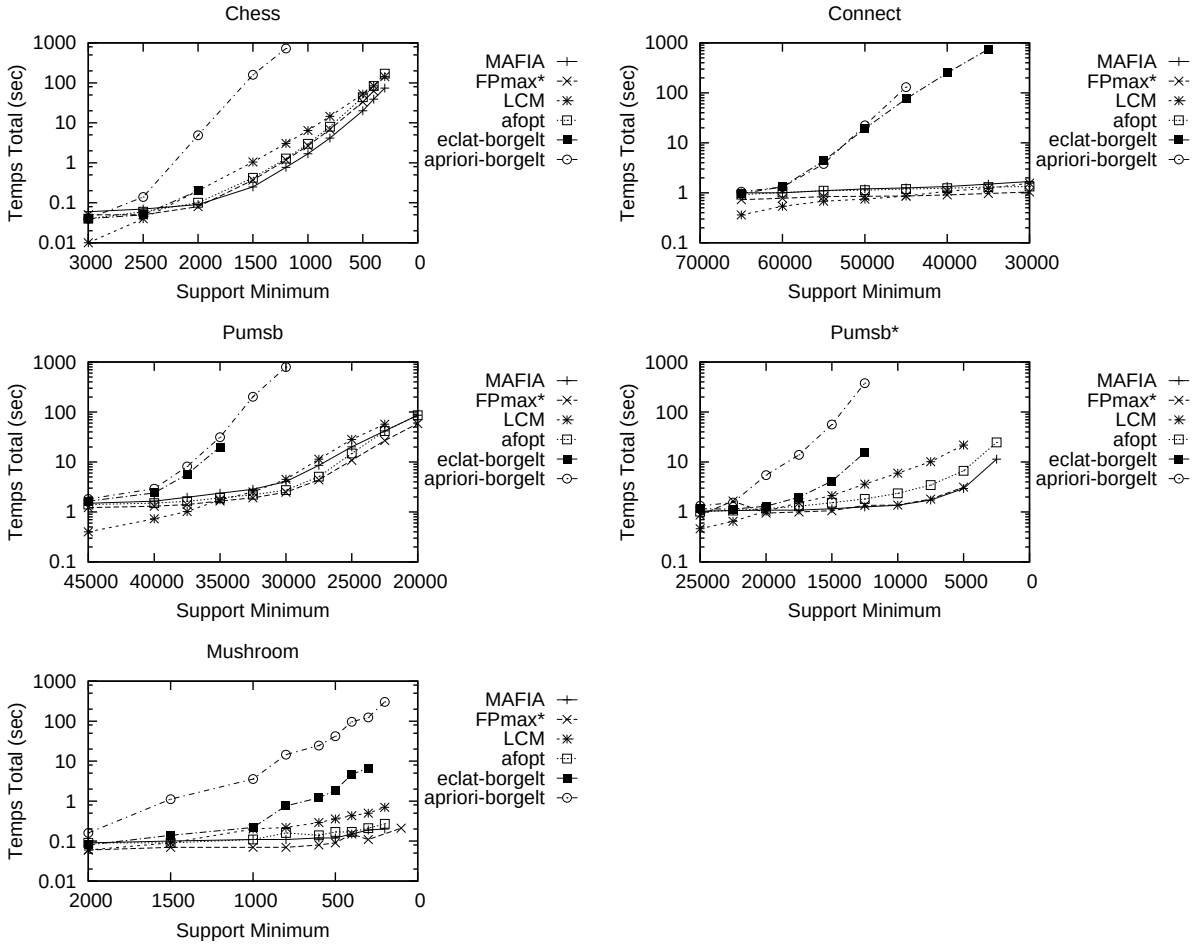

FIG. 5.7 – Performances d'algorithmes de découverte de motifs fréquents maximaux

du critère de densité et les performances des algorithmes.

En effet, les jeux de données *Chess* et *Connect* avec des seuils de support de 30% sont de types différents, i.e. leur densité est différente. Toutefois, les algorithmes ont des performances comparables pour ces jeux de données et ce seuil minimum de support. De plus, notons que *Connect* a plus de transactions et des transactions plus longues que *Chess* (tableau 1). De la même manière, *Pumsb*∗ pour un seuil de support de 10% est plus dense que *Pumsb* pour un seuil de 30% (figures 5.2 et 5.1). Malgré cela, les performances des algorithmes sont encore comparables.

De manière générale, le comportement des algorithmes sur *Connect* et *Pumsb*∗ est relativement efficace et stable. Comme le montre la figure 5.7, leur temps d'exécution pour ces jeux de données augmente très progressivement, et reste faible jusqu'à de petits seuils de support. Au contraire, des jeux tels que *Pumsb* et *Chess* semblent être plus difficiles. Cette remarque est aussi vraie pour des jeux de données tels que *Mushroom*. La majeure partie des algorithmes n'a pas de difficulté sur ce jeu de données même pour des seuils minimums de support très petits (figure 5.7). En comparaison *Chess*, qui est censé être moins dense, est plus difficile.

5.2 Découverte des motifs fréquents maximaux et stratégies adaptatives

Il existe un grand nombre d'algorithmes de découverte des motifs fréquents maximaux [Bay98, LK98, BCG01, GZ01, GZ03, UKA04]. Ces algorithmes ont pour objectif de découvrir la bordure positive des motifs fréquents en limitant l'espace de recherche parcouru grâce à l'anti-monotonie du prédicat "être fréquent". Certains de ces algorithmes ont aussi mis en place des stratégies afin d'adapter la représentation des données et la méthode de comptage du support en fonction des données.

Par exemple, l'algorithme *Mafia* [BCF+03] choisit dynamiquement la structure de données la plus adpatée pour représenter les données. L'algorithme utilise une représentation verticale pour la base de données de transactions. Son principe est d'associer à chaque article l'ensemble des transactions auquel il appartient. Pour faire cela, il utilise deux types de structure de données : un *bitmap* ou une liste d'identifiant de transactions, pour chaque article. Dans le *bitmap*, il y a un bit pour chaque transaction de la base de données. Si l'article i apparaît dans la t-ième transaction, alors le bit t du *bitmap* de l'article i est mis à un ; sinon le bit reste à zéro. Cette structure de données permet d'avoir une procédure de comptage du support des motifs très efficace. Toutefois, la structure peut être creuse pour certains types de données. Dans ces cas, la représentation des données à partir de listes d'identifiant de transactions sera plus intéressante. L'algorithme *Mafia* fait dynamiquement ce choix de structure de données et de techniques de comptage du support en fonction de la proportion d'articles encore utilisés à chaque itération. De plus, lorsqu'un changement de stratégie doit être effectué, l'algorithme reconstruit les

transactions à partir des articles encore utilisés.

LCM [UAUA03] s'appuie aussi sur une représentation verticale des données, et exploite deux techniques pour les stocker et calculer le support. Or, les performances de ces techniques dépendent de la taille des données à traiter. Pour cet algorithme, l'adaptativité se traduit alors par le choix de la solution la plus performante en fonction de la taille de la projection de la base de données considérée.

Il existe globalement trois types de méthodes pour représenter les données et trouver le support des motifs :

- les données sont stockées horizontalement, i.e. chaque transaction est stockée sous la forme d'un ensemble d'articles. Le support d'un motif est compté en faisant des tests d'inclusion dans les transactions ;
- les données sont stockées verticalement, i.e. à chaque article est associée la liste des identifiants des transactions auxquelles il appartient. Le support d'un motif est alors obtenu en faisant l'intersection des listes d'identifiants de transactions des articles qui le composent ;
- les données sont stockées dans des structures de type $FP-tree$ (section 4.2.2).

Pour chacune de ces approches, des structures de données différentes peuvent être utilisées telles que des vecteurs de bits, des listes ou des arbres. Le choix de la structure de données a un fort impact sur la méthode de comptage et sur ses performances, et dépend en grande partie du type de données étudiées. Par exemple, des structures de données telles que les vecteurs de bits seront très efficaces lorsque les données seront denses, mais beaucoup moins dans le cas contraire.

En pratique, la représentation verticale est la plus efficace quand elle est combinée aux algorithmes conservant les projections de la base de données faites à chaque itération. Le problème de cette approche est qu'elle nécessite plus d'espace mémoire.

Pour cette raison, l'algorithme *DCI* [OPPS02] (ou plus récemment $k-DCI$ [OLP+03]) a mis en place une stratégie adaptative qui alterne ces différentes approches en fonction de la quantité de mémoire disponible et des caractéristiques des données traitées. Cet algorithme fait une exploration par niveau de l'espace de recherche. L'approche utilisée pour représenter les données et compter le support dépend tout d'abord de la taille des données et de l'espace mémoire disponible. Si la taille des données est trop importante pour être représentée verticalement, l'algorithme utilise une approche horizontale. A chaque itération, l'algorithme réduit la taille des données. Par conséquent, lorsqu'elles sont assez petites pour tenir en mémoire, l'algorithme change de stratégie, et utilise une approche verticale. Ensuite, l'algorithme applique différentes optimisations en fonction de la "densité" des données étudiées. Cette densité mesure la corrélation entre les listes de transactions et les articles les plus fréquents. Elle est obtenue en recherchant le nombre maximum d'articles dont l'intersection de leur liste de transactions est significative. Par exemple, si $1/4$ des articles partagent 90% des transactions, la densité sera égale à $1/4 \times 90\% = 0.23$. Si cette valeur est supérieure à un seuil fixé, la projection de la base de données traitée sera

considérée comme dense.

Notons que les aspects adaptifs sont très développés dans cet algorithme. En général, une seule représentation des données est considérée, seules les structures de données et techniques de comptage sont adaptées au type de données. Par exemple, $Mafia$ [BCF+03] et LCM [UAUA03] utilisent toujours une représentation verticale de la base de données, mais adaptent dynamiquement leurs techniques de comptage du support et de stockage des données. L'algorithme $Afopt$ [LLY+03] qui est un algorithme du type "$FP-growth$", adapte uniquement les structures de données utilisées pour représenter le $FP-tree$ (section 4.2.2). En effet, les performances de ce type d'algorithme dépendent en grande partie du coût de construction et de parcours de chacun des $FP-tree$ construit lors du parcours en profondeur. $Afopt$ détermine la représentation à utiliser en fonction du nombre d'articles fréquents considérés dans le $FP-tree$, de leur support moyen et du seuil minimum de support.

Toutefois, les aspects adaptatifs mis en place sont focalisés sur la gestion des données, i.e. trouver la représentation et/ou la technique de comptage la plus appropriée aux données en cours de traitement. Or, ces stratégies adaptatives de gestion des données ne suffisent pas. Les résultats obtenus lors des ateliers FIMI ont confirmé cela, car aucune des implémentations n'était la meilleure pour tous les jeux de données malgré ces stratégies adaptatives. Face à ce problème, le développement de stratégies adaptant dynamiquement leur parcours de l'espace de recherche en fonction des données est une alternative intéressante.

5.3 Autres applications des caractéristiques des jeux de données

D'autres travaux tels que [RMZ03, POP04, RZM05] ont montré que les observations issues de l'étude des jeux de données pouvaient être aussi utiles dans beaucoup d'autres domaines, de la prédiction de performance, la détermination du seuil minimum de support, l'échantillonnage, à la génération de jeux de données synthétiques.

Dans [RMZ03, RZM05], la distribution de la bordure positive est considérée comme un élément clé pour caractériser les bases de données de transactions. Les auteurs ont prouvé que toutes les distributions sont possibles, et peuvent être rencontrées en pratique. De plus, ils proposent une méthode pour générer des bases de données synthétiques à partir d'une distribution de bordure positive en entrée.

Toujours à partir du travail effectué dans [GZ01], [POP04] a proposé une mesure pour caractériser la densité des jeux de données. Plus concrètement, ils considèrent la base de données de transactions comme une source d'information et mesure l'entropie du signal, i.e. la quantité d'information produite par cette source. Les auteurs ont aussi montré comment une telle caractéristique pouvait être utile dans un grand nombre de domaines. Par

exemple, cette mesure peut être utilisée pour prédire le nombre de motifs fréquents à un seuil minimum de support donné, ce qui permet d'estimer le temps d'exécution des algorithmes à de petits seuils de support sans avoir à les exécuter. Elle peut aussi être utilisée pour estimer si un échantillon d'un jeu de données contient la même quantité d'informations que le jeu de données initial. Si c'est le cas, l'échantillon pourra être utilisé par un algorithme à la place du jeu de données initial, et permettra d'obtenir une approximation du résultat.

Cependant, ces travaux se focalisent principalement sur la bordure positive. Or, comme nous venons de le voir, ce critère ne suffit pas pour caractériser un jeu de données.

Chapitre 6

Une nouvelle caractérisation et classification des jeux de données

Sommaire

Jusqu'à présent, la caractérisation d'un jeu de données était faite à partir des caractéristiques des transactions et parfois de la distribution de la bordure positive. Comme nous venons de le voir, ces critères sont nécessaires mais ne permettent pas toujours de bien caractériser le type de jeux de données étudiés par les algorithmes.

Dans cette partie, nous allons présenter une étude expérimentale poussée des jeux de données de FIMI. La nouveauté réside dans l'étude de la bordure négative qui, en complément de la bordure positive, permet d'avoir une caractérisation plus fine des jeux de données. Nous présenterons aussi comment cette caractéristique permet d'expliquer en grande partie les performances des algorithmes, et comment elle permet d'obtenir une nouvelle classification des jeux de données.

A notre connaissance, cette étude est la première à s'intéresser à la compréhension des jeux de données pour les motifs fréquents en utilisant leur bordure négative.

6.1 Etude expérimentale des jeux de données

6.1.1 Protocole expérimental

Les expérimentations ont été effectuées sur les quatorze jeux de données représentatifs utilisés lors des ateliers FIMI [Goeb]. Chaque jeu de données a été étudié pour plusieurs seuils minimums de support, de seuils très élevés à des seuils très petits. Pour chaque seuil, les motifs fréquents, fermés fréquents, générateurs fréquents, et essentiels fréquents ont été recherchés. Nous avons étudié la distribution de leurs motifs par rapport à leur taille, i.e. le nombre de motifs par niveau (de 1 à la taille des plus grands motifs). De plus, nous avons étudié la distribution de la bordure positive et de la bordure négative des motifs fréquents, générateurs fréquents, et essentiels fréquents. Comme nous l'avons vu au chapitre 2, l'ensemble des fermés n'est pas fermé par le bas, et donc la notion de bordure ne peut être appliquée dans ce cas.

Pour mener ces expérimentations, nous avons utilisé des implémentations d'algorithmes disponibles sur le site du FIMI [Goeb]. La découverte des motifs fréquents, des motifs fréquents maximaux et des fermés fréquents a été effectuée à partir des algorithmes $FPClose$ et $FP-growth*$ de [GZ03]. L'implémentation de l'algorithme *Apriori* de [Bor03] a été adaptée pour découvrir la bordure négative des motifs fréquents, ainsi que pour découvrir les motifs générateurs fréquents, essentiels fréquents et leurs bordures.

6.1.2 Résultats expérimentaux

Afin de réaliser une comparaison avec les résultats de [GZ01], nous présenterons principalement les résultats expérimentaux obtenus sur les jeux de données $Chess$, $Pumsb$, $Connect$, $Pumsb*$, $Mushroom$, et $T10I4D100K$. Les résultats expérimentaux détaillés pour tous les jeux de données de FIMI sont décrits dans [Flo05].

Dans la suite de cette section et dans les différents schémas, nous utiliserons les notations suivantes :

F	motifs fréquents
$FClosed$	motifs fermés fréquents
$FGen$	motifs générateurs fréquents
$FEss$	motifs essentiels fréquents

Le tableau 6.1 montre un échantillon des résultats obtenus par nos expérimentations. Par exemple, pour un seuil minimum de 30%, $Chess$ a 59589 motifs fréquents de taille 4, 13221 motifs fermés fréquents de taille 4, 44096 motifs générateurs fréquents de taille 4 et 62383 motifs essentiels fréquents de taille 4.

Nous allons étudier nos résultats expérimentaux par rapport à trois principaux aspects :

- la distribution des bordures des motifs fréquents,
- la distribution des bordures des représentations condensées,
- la stabilité des distributions relativement aux variations de support.

Itemset size	F	$FClosed$	$FGen$	$FEss$	F		$FGen$		$FEss$	
					Bd^-	Bd^+	Bd^-	Bd^+	Bd^-	Bd^+
1	50	27	50	50	25		25	1	25	1
2	896	338	828	828	329		397	2	397	149
3	9049	2568	7628	4240	928	1	988	34	4376	853
4	59589	13221	44096	6283	3371	9	3440	268	8519	3178
5	273069	49002	170161	1635	10118	89	10178	1343	1764	1186
6	907800	137564	456826	116	21405	439	21416	4876	36	109
7	2255159	303661	875938	1	33711	1369	33720	11963		1
8	4276852	540861	1216501		39910	3686	39910	22521		
9	6291848	787143	1231162		33890	8200	33890	31137		
10	7263312	940504	903996		21894	14804	21894	32243		
11	6626801	923310	474618		10160	21183	10160	25491		
12	4790827	740773	172688		3507	24638	3507	15326		
13	2738089	481499	41186		791	23766	791	6403		
14	1227702	250715	5787		114	18088	114	1951		
15	425896	102977	360		8	10934	8	314		
16	111726	32875	3			5085		3		
17	21328	7908				1734				
18	2757	1370				496				
19	206	145				97				
20	6	6				6				
total	37282962	5316467	5601828	13153	180161	134624	180438	153876	15117	5477

TAB. 6.1 – *Chess* dataset, $minsup = 30\%$

Distribution des bordures des motifs fréquents

Considérons la bordure positive et négative des motifs fréquents des six jeux de données considérés (figure 6.1).

Etudions la distribution d'une bordure par rapport à l'autre, et notamment la "distance" entre les deux bordures. Intuitivement, chaque distribution permet de situer l'emplacement des motifs correspondants dans l'espace de recherche. Ainsi, l'étude des distributions des bordures positive et négative permet de situer globalement où sont ces deux bordures l'une par rapport à l'autre dans l'espace de recherche. Dans ce contexte, la distance entre ces distributions représente l'espace de recherche "séparant" les deux bordures. Cette partie de l'espace de recherche est particulièrement intéressante car d'un point de vue théorique, l'espace de recherche est entièrement caractérisé dès la découverte de l'une des deux bordures (section 1).

De manière générale, nous observons deux comportements différents par rapport à la "distance" (notée d) entre les deux bordures. Pour *Chess* et *Pumsb* (figure 6.1), les distributions des deux bordures sont très proches ($d = 4$), i.e. le sommet de la courbe de la bordure négative (8) est seulement quelques niveaux avant celui de la bordure positive (12). En revanche, les jeux de données *Connect*, *Pumsb*$*$ et *Mushroom* sont différents : la distance entre leurs deux bordures est importante ($d = 14$ pour *Connect*, $d = 12$ pour *Pumsb*$*$, et $d = 14$ pour *Muhroom*). Le jeu de données $T10I4D100K$ a lui aussi une

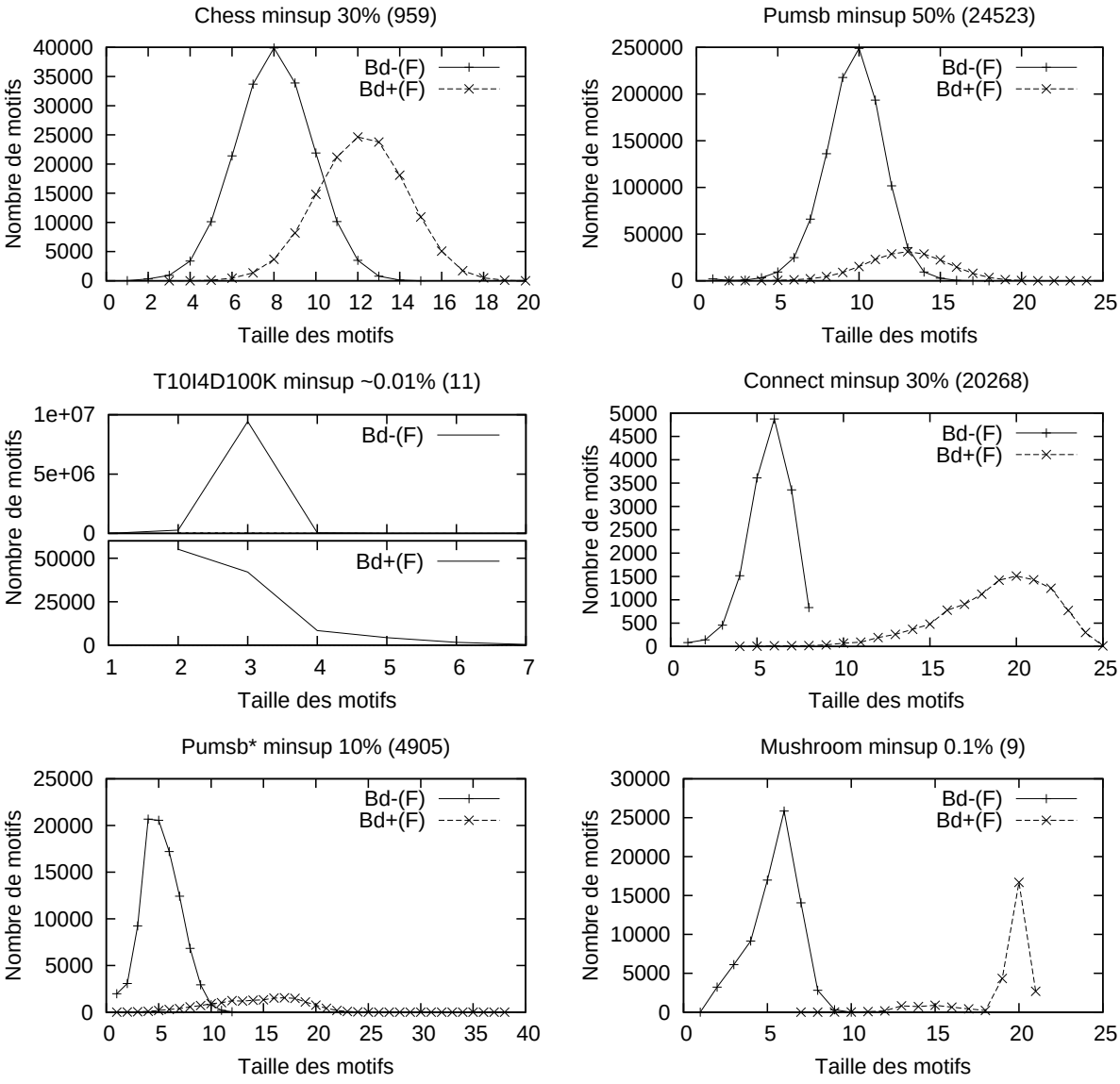

FIG. 6.1 – Bordures des motifs fréquents

faible distance entre la bordure négative et la bordure positive. Toutefois, il est différent de *Chess* et *Pumsb* puisque ces bordures sont composées de motifs de petite taille.

Distribution des bordures des représentations condensées

Considérons maintenant les bordures des motifs générateurs fréquents (figure 6.2) et essentiels fréquents (figure 6.3) pour les jeux de données *Chess* et *Connect*.

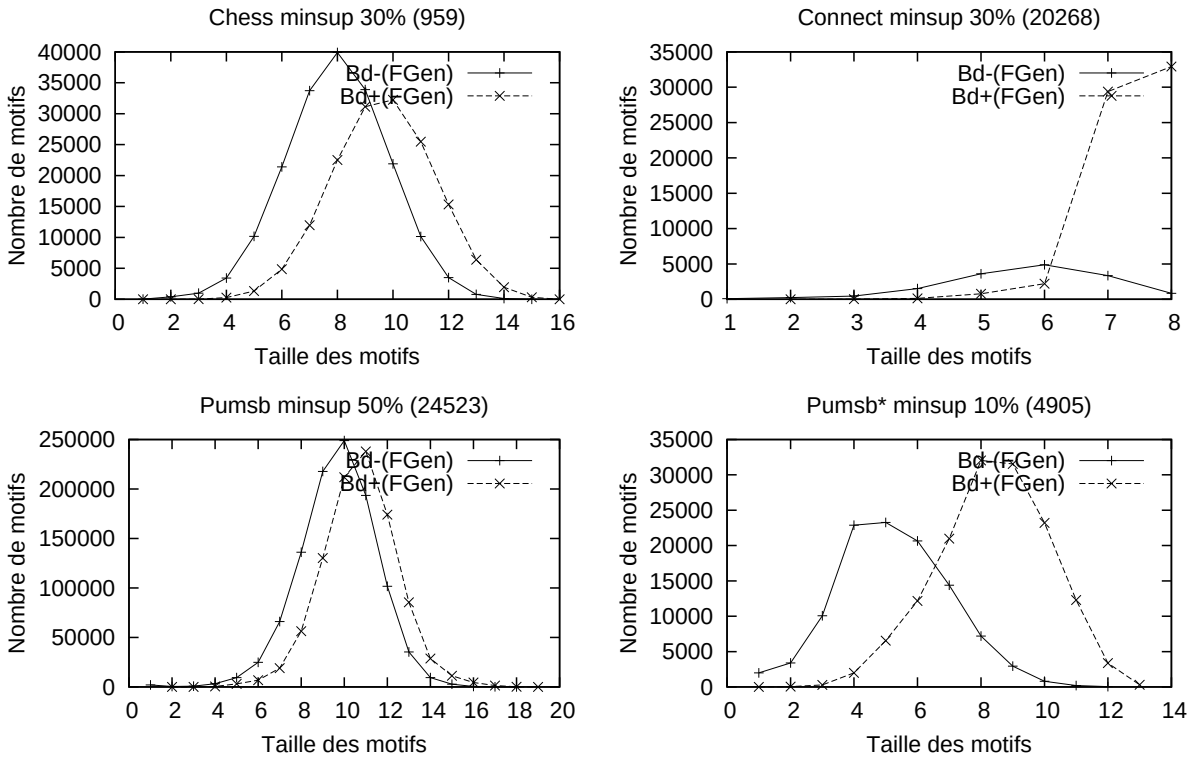

FIG. 6.2 – Bordures des motifs générateurs fréquents

Comme le montrent les figures 6.2 et 6.3, la distribution des deux bordures ressemble encore à une "courbe en cloche". Ce comportement étant aussi observé pour les bordures des motifs fréquents, ce type de courbe semble donc indépendant du prédicat anti-monotone considéré. De plus, la distance entre la bordure négative et la bordure positive est petite pour chacune de ces représentations condensées.

Le même comportement a été constaté pour toutes nos expérimentations [Flo05].

Stabilité des distributions

Nous étudions maintenant la variation du seuil minimum de support par rapport à la distribution des bordures des motifs fréquents. Par exemple, considérons les jeux

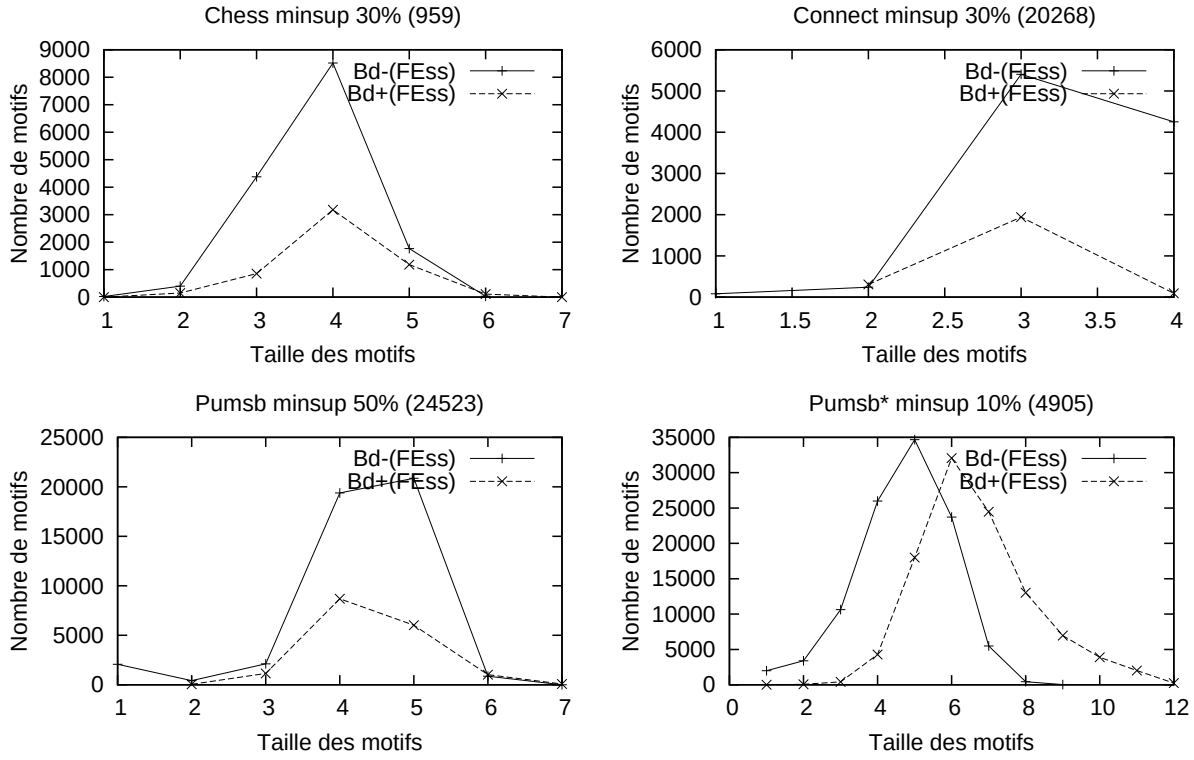

FIG. 6.3 – Bordures des motifs essentiels fréquents

de données *Chess*, *Connect*, *Pumsb*, et *Pumsb*∗ pour des seuils minimums de support différents. La figure 6.4 représente sur une même ligne les distributions des bordures de chacun des jeux de données étudiés pour trois seuils minimums de supports différents.

De manière surprenante, la position relative des bordures des motifs fréquents apparaît relativement *stable* par rapport à la variation du seuil minimum de support. Par exemple, la distance entre les deux bordures des motifs fréquents est toujours importante pour *Connect*, et toujours faible pour *Chess* (figure 6.4, deux premières lignes). Cette observation suggère une sorte de structure globale des données peu sensible aux variations du seuil minimum de support.

Notons qu'en apparence la distribution de la bordure positive de *Mushroom* semble moins stable que les autres. Toutefois, en étudiant un plus grand nombre de seuils de support [Flo05], il apparaît que sa distribution évolue très régulièrement. La distribution de la bordure positive est toujours constituée de deux "pics", i.e. deux augmentations du nombre de motifs suivi d'une diminution. Ces deux pics augmentent tous deux progressivement avec la baisse du seuil de support, mais avec une intensité différente. Le nombre de grands motifs fréquents (le second "pic") est de plus en plus important avec la diminution du seuil de support. De plus, la bordure négative est toujours plus "basse" que la bordure positive.

De manière générale, cette stabilité a été observée pour toutes nos expérimenta-

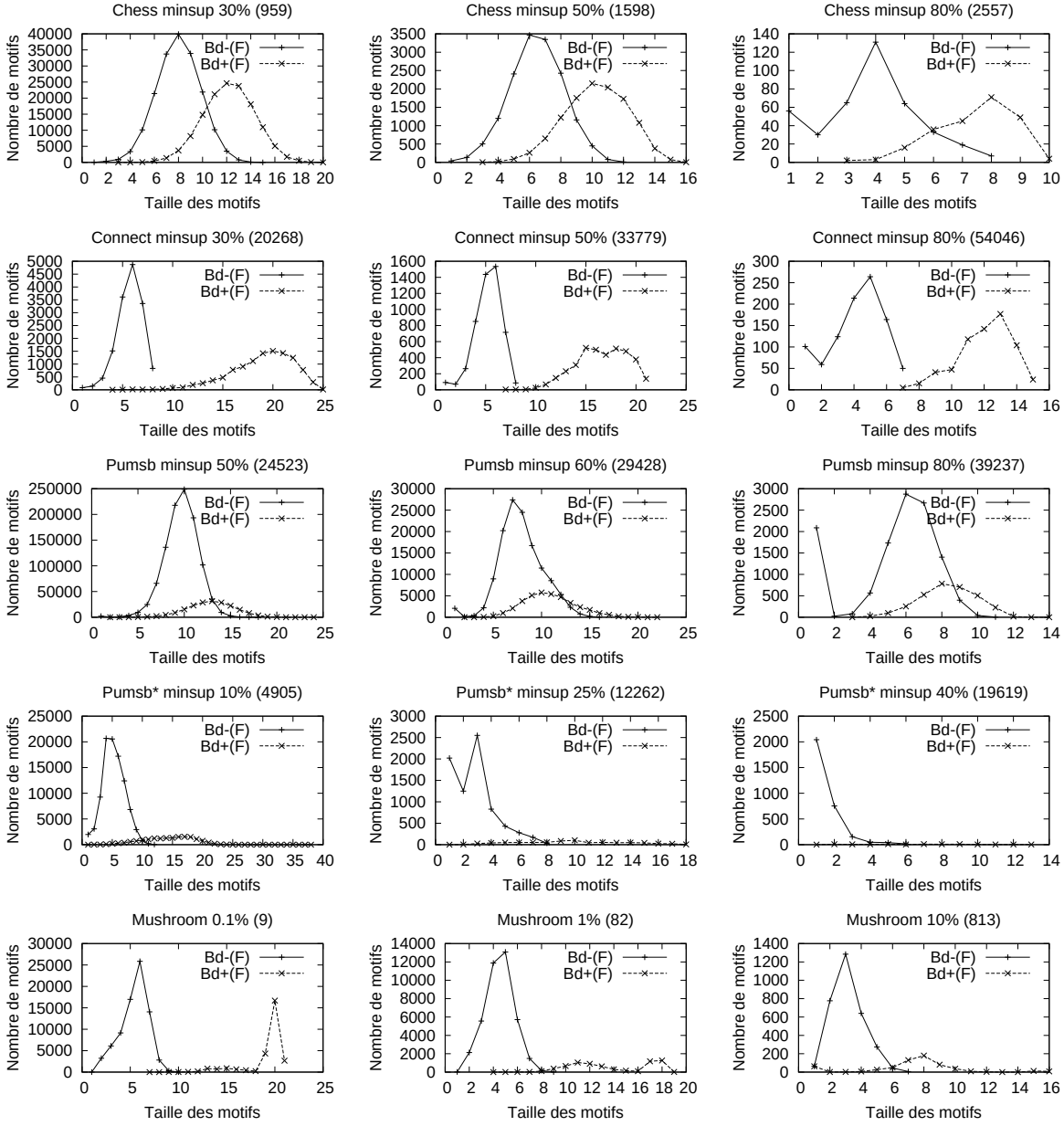

FIG. 6.4 – Bordures des motifs fréquents pour des seuils minimums de support différents

tions [Flo05]. De plus, il apparaît que les distributions des autres ensembles étudiés (motifs fréquents, fermés fréquents, générateurs fréquents, essentiels fréquents et leurs bordures) sont aussi relativement stables les unes par rapport aux autres.

6.2 Impact des bordures sur les algorithmes

Dans cette section, nous allons étudier l'influence de la distribution des bordures sur les stratégies d'exploration de l'espace de recherche présentées en chapitre 4.

Nous allons plus particulièrement nous focaliser sur les algorithmes de découverte des motifs fréquents maximaux, et allons étudier les performances de leur implémentation [BGZ04]. L'intérêt de ce type d'algorithmes est de se concentrer uniquement sur l'exploration de l'espace de recherche, et non sur la génération et le comptage de l'ensemble des motifs fréquents comme le font les algorithmes de découverte des motifs fréquents.

6.2.1 Performances et bordures

La figure 6.5 représente à gauche les temps d'exécution des algorithmes, et à droite les distributions des bordures des motifs fréquents des jeux de données considérés. Notons que l'axe des ordonnées représentant les temps d'exécution a une échelle logarithmique.

De manière générale, on peut déduire de la figure 6.5 que la position des bordures dans l'espace de recherche et la distance entre celles-ci influencent les performances des algorithmes. En effet, les algorithmes sont plus performants lorsque la bordure négative est "basse" dans l'espace de recherche (i.e. composée de petits motifs). Toutefois, lorsque la bordure positive est constituée de grands motifs, les algorithmes "classiques" tels que *Apriori* et *Eclat* sont tout de même mis en échec. L'efficacité des autres algorithmes dépend de la "distance" entre les deux bordures : plus la distance entre les bordures des motifs fréquents est importante, plus les algorithmes sont performants.

Pour le jeu de données *Chess* (figure 6.5, première ligne), les temps d'exécution des algorithmes augmentent de façon exponentielle avec la diminution du seuil minimum de support. A l'opposé, pour *Connect* (quatrième ligne), ils apparaissent presque linéaires pour la majeure partie des algorithmes ($Mafia$ [BCF+03], $FPmax*$ [GZ03], LCM [UKA04] et $afopt$ [LLY+03]). De plus, il est intéressant de rappeler que *Connect* a un nombre de transactions plus important et des transactions plus longues que *Chess* (tableau 1).

Le même type de comportement peut être observé pour des jeux de données tels que *Pumsb* et *Pumsb*$*$ (figure 6.5). Ces deux jeux de données sont très similaires par rapport au type d'informations qu'ils contiennent, au nombre d'articles, au nombre et à la taille des transactions. Pourtant, les algorithmes restent efficaces pour des seuils très petits avec de longs motifs fréquents pour *Pumsb*$*$, alors que pour *Pumsb* les algorithmes n'ont pas

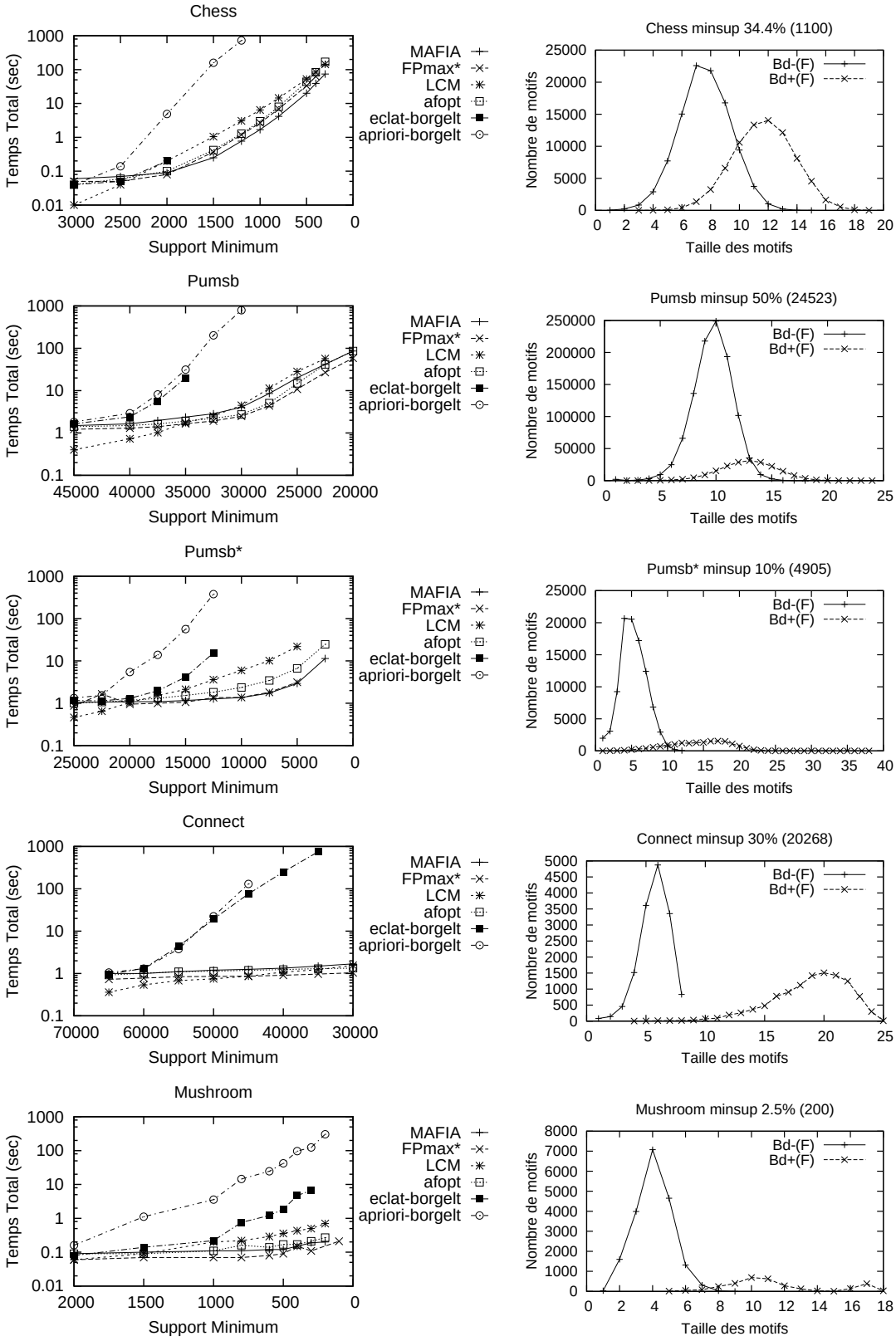

FIG. 6.5 – Performances des algorithmes et bordures des motifs fréquents

de bonnes performances, même pour des seuils relativement élevés.

De plus, comme le montre le tableau 6.2, le nombre de motifs fréquents des jeux de données *Connect* et *Pumsb*∗ pour ces seuils de support est beaucoup plus important que celui de *Chess* et *Pumsb*. Or le nombre de motifs fréquents permet d'estimer la taille de l'espace de recherche à parcourir (i.e. le nombre de motifs à tester) dans le pire cas. Cette remarque est intéressante car on aurait pu penser que cette différence était liée à un nombre moins important de motifs fréquents pour ces jeux, ce qui n'est donc pas le cas, bien au contraire.

Pour *Mushroom* (2.5%) et *Chess* (34.4%), la taille des motifs fréquents maximaux (figure 6.5) et le nombre de motifs fréquents (tableau 6.2) sont relativement identiques. Toutefois, les algorithmes sont plus efficaces avec *Mushroom*.

	Nombre de motifs fréquents
Chess minsup 34.4%	16 763 342
Connect minsup 30%	1 331 673 367
Pumsb minsup 50%	$\approx 1,65 \times 10^8$
Pumsb∗ minsup 10%	$\approx 5,5 \times 10^{11}$
Mushroom minsup 2.5%	18 094 857

TAB. 6.2 – Nombre de motifs fréquents pour les jeux de données et seuils minimums de support étudiés

6.2.2 Analyse des résultats

Expérimentalement, nous avons constaté l'influence des bordures des motifs fréquents sur les algorithmes. Nous allons maintenant étudier les stratégies employées par ces algorithmes et l'influence de la distance entre les bordures sur ces stratégies.

Pour les algorithmes "classiques" tels que *Apriori* ou *Eclat*, les performances sont conditionnées en grande partie par la distribution de la bordure positive. En effet, ces algorithmes doivent parcourir l'ensemble des motifs couverts par la bordure positive, i.e inclus dans un motif de la bordure positive. Donc plus les motifs de la bordure positive sont grands, plus ces algorithmes ont des difficultés. Toutefois, comme nous l'avons observé précédemment, ces algorithmes sont aussi influencés par la distribution de la bordure négative. Les motifs de cette bordure sont utilisés pour élaguer l'espace de recherche. Par conséquent, les découvrir rapidement signifie élaguer rapidement de grandes parties de l'espace de recherche.

Les autres algorithmes exploitent en plus les motifs fréquents maximaux pour élaguer l'espace de recherche, et comme nous l'avons vu, sont plus particulièrement influencés par la distance entre les deux bordures. Pour en comprendre la raison, reprenons l'exemple 3. La bordure positive des motifs fréquents est composée de $\{ABCDF, BDEF\}$ et la bordure négative correspondante de $\{AE, CE\}$. Les bordures sont donc éloignées. Etudions

maintenant le comportement des algorithmes $Mafia$, $FPmax*$ et LCM lors de la découverte des motifs fréquents maximaux. Les figures 6.6, 6.7, et 6.8 suivantes représentent ce parcours. Les motifs noirs sont les motifs fréquents testés, en gras ce sont les motifs fréquents maximaux. Les motifs hachurés sont les motifs non fréquents testés. Les motifs en gris sont ceux qui n'ont pas été générés grâce aux motifs fréquents maximaux découverts.

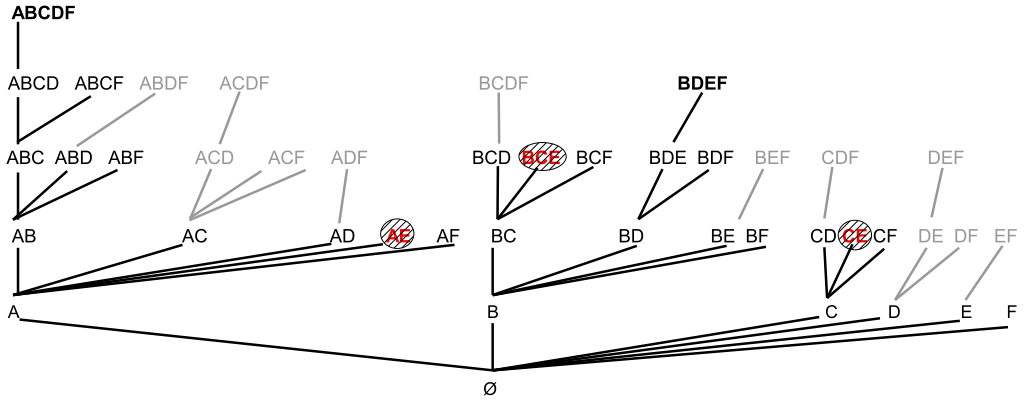

FIG. 6.6 – Parcours de l'espace de recherche effectué par $Mafia$

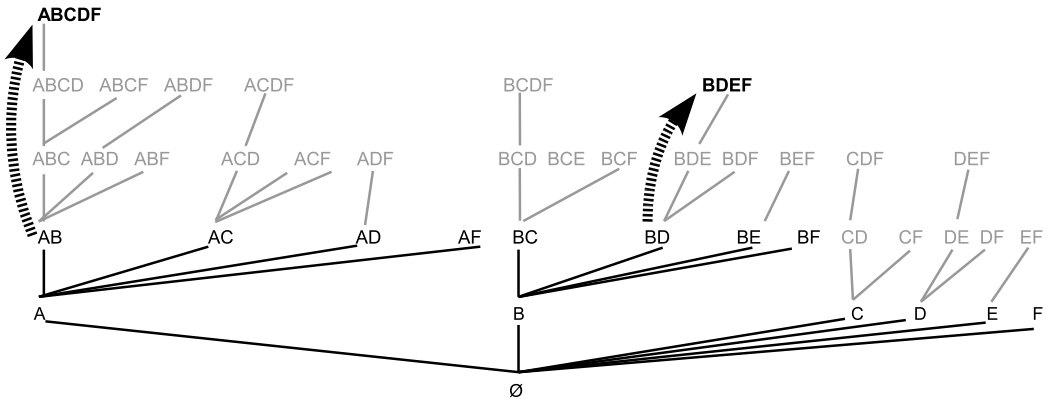

FIG. 6.7 – Parcours de l'espace de recherche effectué par $FPmax*$

D'après ces figures, la différence de taille entre les motifs de la bordure positive et ceux de la bordure négative se traduit par l'élagage par les algorithmes de grandes parties de l'espace de recherche. Par exemple, pour $Mafia$ (figure 6.6), la découverte de $ABCDF$ permet d'élaguer les motifs $\{ABDF, ACDF, BCDF, ABDF, ACD, ACF, ADF, CDF\}$ et la découverte de $BDEF$ permet d'élaguer les motifs $\{BEF, DEF, DE, DF, EF\}$. Notons que $FPmax*$ (figure 6.7) élague encore plus de motifs grâce aux sauts dans l'espace de recherche qu'il effectue.

Prenons par exemple, le cas extrême où une grande partie des motifs de la bordure négative sont un niveau après ceux de la bordure positive. Supposons que la bordure positive soit composée de $\{ABCD, ABCE, ABCF, ABDE, ABDF, ABEF, ACDE, ACDF, ACEF, ADEF,$

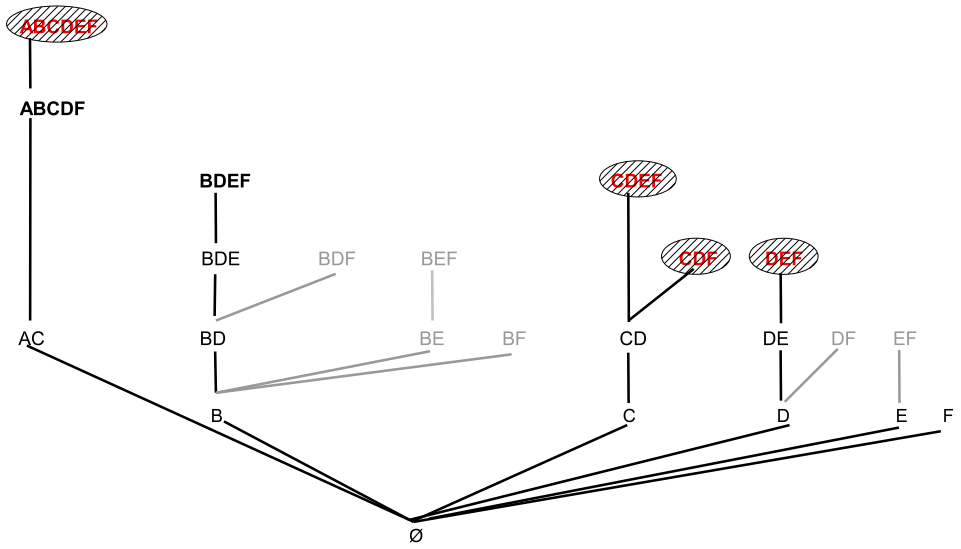

FIG. 6.8 – Parcours de l'espace de recherche effectué par LCM

$BCDE, BCDF, BCEF, BDEF\}$, et la bordure négative de $\{ABCDE, ABCDF, ABCEF, ABDEF, CDEF\}$. En conservant les mêmes notations que précédemment, $Mafia$ effectuera donc le parcours illustré par la figure 6.9.

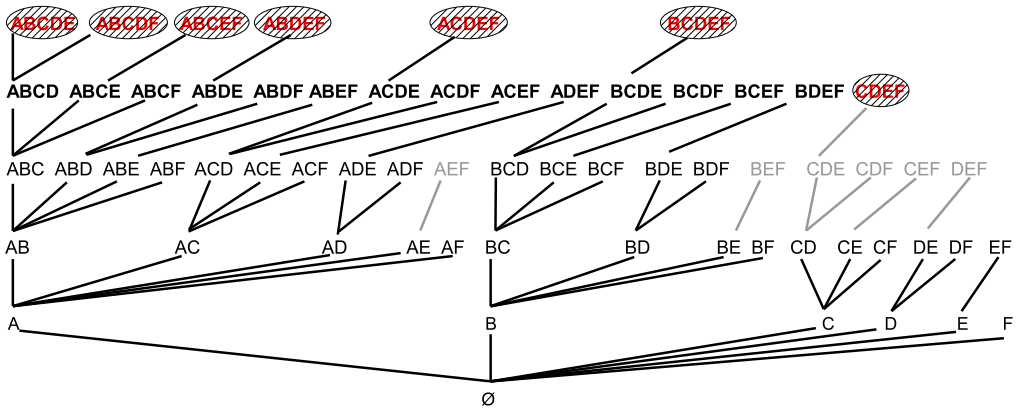

FIG. 6.9 – Parcours de l'espace de recherche effectué par $Mafia$ pour des bordures proches

Comme on peut le constater sur cette figure, le nombre de motifs générés et testés est beaucoup plus important que lorsque les bordures sont éloignées (figure 6.6). En effet, lorsque la bordure négative est découverte, l'espace de recherche des fréquents est alors caractérisé, et la bordure positive peut théoriquement être déduite directement (d'après le théorème 2 introduit en section 3). Par conséquent, plus la bordure négative est découverte, plus l'espace de recherche des fréquents est caractérisé et plus les motifs de la bordure positive pourront être déduits.

L'algorithme LCM est différent des autres algorithmes car il effectue son parcours de l'espace de recherche en s'appuyant sur les motifs fermés. De par cette stratégie, ce

type d'algorithme ne teste donc pas les motifs d'une même classe d'équivalence, mais uniquement leur fermé. Intuitivement, l'algorithme se comporte comme s'il faisait des "sauts" entre les fermés des différentes classes d'équivalence. Cette stratégie sera donc plus particulièrement efficace lorsqu'il y aura de grandes classes d'équivalence. Or d'un point de vue théorique, une grande distance entre les bordures des motifs fréquents ne signifie pas nécessairement qu'il y aura de grandes classes d'équivalence. Prenons par exemple le cas où la base de données est composée de l'ensemble des parties des motifs de la bordure positive, les classes d'équivalence seront toutes composées d'un seul motif, et l'algorithme devra donc potentiellement parcourir l'ensemble des motifs fréquents.

Toutefois d'après les expérimentations, il apparaît que lorsque la distance entre les bordures des motifs fréquents est importante, il y a de grandes classes d'équivalence. Pour observer cela, nous étudions les distributions des motifs fermés fréquents par rapport aux motifs générateurs fréquents. Pour rappel, ces derniers sont les plus petits motifs des classes d'équivalence étudiées. Par conséquent, l'étude de la distance entre ces deux distributions permettra de détecter l'existence de grandes classes d'équivalence.

La figure 6.10 représente la distribution des motifs générateurs fréquents et fermés fréquents pour les jeux de données étudiés précédemment. Comme on peut le constater, pour des jeux de données tels que *Connect*, la distance entre les distributions des motifs générateurs fréquents et des fermés fréquents est aussi relativement importante. Notons toutefois qu'elle est moins grande que pour les distributions des bordures des motifs fréquents.

Les observations décrites dans cette section nous conduisent à déterminer une nouvelle classification des jeux de données par rapport à la distribution des bordures.

6.3 Vers une nouvelle classification des jeux de données

Notre nouvelle classification des jeux de données diffère de celle proposée dans [GZ01] car elle prend en compte la bordure négative des motifs fréquents en plus de la bordure positive.

En nous appuyant sur les distributions des deux bordures des motifs fréquents et sur la "distance" entre celles-ci, nous avons classé les jeux de données en trois types :

- Type I : jeux de données où les distributions des bordures sont très proches, i.e. la courbe de la bordure négative n'est pas éloignée de celle de la bordure positive. Autrement dit, la majeure partie des motifs dans les deux bordures ont approximativement la même taille. *Chess* et *Pumsb* sont par exemple ce type de jeux de données. De tels jeux de données vont poser des difficultés aux algorithmes de découverte des motifs fréquents.
- Type II : jeux de données où une grande distance sépare les deux bordures. Les motifs de la bordure négative sont beaucoup plus petits que ceux de la bordure

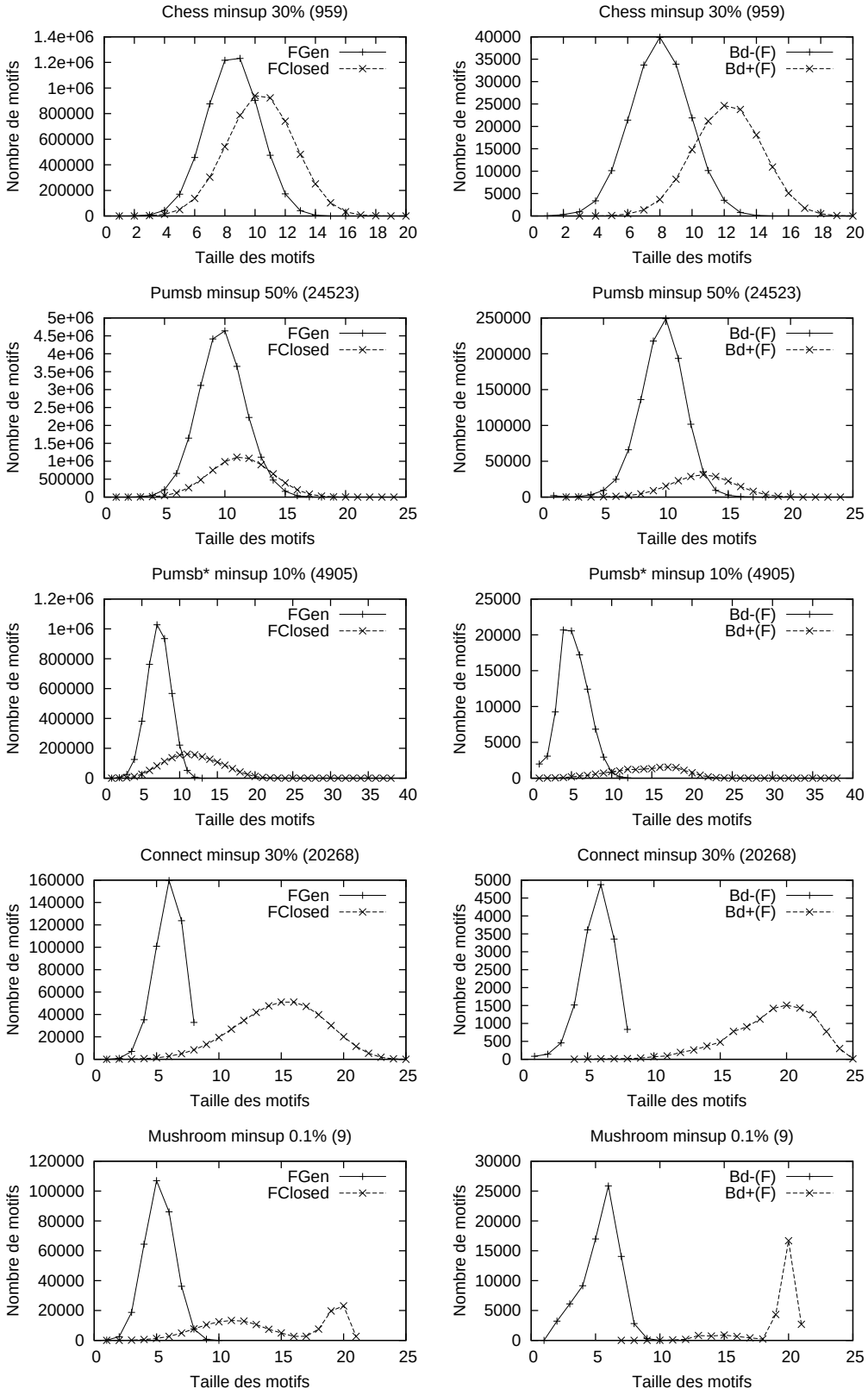

FIG. 6.10 – Distribution de $FClosed$, $FGen$, $\mathcal{B}d^{+}(F)$ et $\mathcal{B}d^{-}(F)$

positive. *Connect* et *Pumsb*∗ sont dans cette catégorie. En pratique, ce type de jeux de données est plus facile que le précédent.
- Type III : cas particulier de jeux de données de type I. Les deux distributions sont très proches, mais situées dans les niveaux les plus bas de l'espace de recherche. Ces jeux de données correspondent aux données éparses. $T10I4D100K$ est un exemple de ce type de jeux de données. En pratique, ce sont souvent les jeux de données les plus simples pour les algorithmes jusqu'à des seuils minimums de support très petits (<1%). En dessous de ces seuils, l'extraction devient difficile en raison du grand nombre de motifs fréquents.

Le tableau 6.3 résume cette nouvelle classification et positionne chacun des jeux de données de FIMI par rapport à celle-ci.

Type	Type I	Type II	Type III
Temps d'exécution des algorithmes (observés pendant FIMI'03 et FIMI'04)	Elevé	Faible	Faible pour la majorité des seuils Elevé pour des seuils très petits (<1%)
"Distance" entre les bordures	petite	grande	petite
Taille des motifs	grands	grands	petits
Exemples de jeux de données	*Chess*, *Pumsb*, *Accidents*, *Webdocs*	*Connect*, *Pumsb*∗, *Mushroom*	$T10I4D100K, BMS-WebView1$, $BMS-WebView2, BMS-Pos$, $Kosarak, retail, T40I10D100K$

TAB. 6.3 – Nouvelle classification des jeux de données

Cette classification est plus simple que celle présentée dans [GZ01] tout en étant stable par rapport au changement de seuil minimum de support (figure 6.4). En plus des caractéristiques habituellement étudiées, la "distance" entre les distributions de la bordure négative et de la bordure positive permet de mieux évaluer la "difficulté" d'un jeu de données pour les algorithmes.

Pour résumer, les principaux avantages de la classification proposée sont :
- une meilleure correspondance entre la classification et les performances des algorithmes. En d'autres termes, cette classification est une première étape vers l'évaluation de la "difficulté" d'un jeu de données pour les algorithmes ;
- la stabilité de la classification par rapport au changement de seuil minimum de support.

Toutefois, les caractéristiques des bases de données et la taille des motifs de la bordure positive doivent être aussi considérées. Elles permettent notamment d'expliquer l'efficacité des algorithmes sur *Chess* pour des seuils minimums de support élevés, celui-ci ayant moins d'articles, moins de transactions et des transactions plus petites que les autres jeux de données.

Chapitre 7

ABS : un algorithme adaptatif de découverte des bordures des motifs fréquents

Sommaire

A partir de l'étude des jeux de données effectuée, nous allons présenter dans ce chapitre un algorithme adaptatif pour la découverte des bordures des motifs fréquents. Cet algorithme, appelé *ABS* (*Adaptive Borders Search*) [FMP04], est dérivé de l'algorithme *ZigZag* [MP03] développé au sein de l'équipe pour la découverte des DI satisfaites dans une base de données.

7.1 Principes de l'algorithme

ABS est une approche hybride qui change de stratégie au cours de l'exploration de l'espace de recherche. Il effectue dans un premier temps un parcours par niveau, puis

alterne des sauts ou dualisations entre les deux bordures. Ce changement de stratégie se fait dynamiquement en fonction des données.

7.1.1 Un début de parcours par niveau

La première phase de l'algorithme consiste à utiliser l'algorithme *Apriori* [AS94] (section 4.1) pour explorer par niveau l'espace de recherche, des motifs les plus petits aux plus grands, juqu'à un niveau k fixé dynamiquement à partir des données. Cette approche permet ainsi de trouver les motifs de la bordure positive de taille strictement inférieure à k, notée $\mathcal{B}d^+_{\leq\ k-1}(F)$, ainsi que les motifs de la bordure négative de taille inférieure ou égale à k, notée $\mathcal{B}d^-_{\leq\ k}(F)$.

Un facteur justifie principalement ce début de parcours de l'espace de recherche. En pratique, le nombre de motifs non fréquents de petite taille peut être très élevé. Par exemple, le jeu de données $T10I4D100K$ (figure 6.1) peut avoir près de dix millions de motifs non fréquents minimaux de taille inférieure à trois. Dans ce cas, une approche par niveau permet donc d'élaguer très rapidement une grande partie de l'espace de recherche.

De plus, les bancs d'essais réalisés lors des ateliers FIMI [BZ03, BGZ04] ont montré qu'une approche par niveau était très efficace lorsque les motifs fréquents étaient de petite taille. Les performances de l'implémentation d'*Apriori* testée [Bor03] égalaient même les meilleures implémentations sur ce type de jeux de données.

Notre objectif est donc d'exploiter l'efficacité d'*Apriori*, et de changer de stratégie lorsque celui-ci est mis en difficulté. Ensuite, la connaissance découverte lors de ce parcours est exploitée pour débuter une stratégie différente basée sur la notion de dualisation (section 4.3).

7.1.2 De la bordure négative à la bordure positive

Une fois l'approche par niveau stoppée, l'algorithme change de stratégie, et effectue des "sauts" dans l'espace de recherche. Ces sauts s'appuient sur les informations déjà connues pour estimer la bordure positive. Plus précisément, les motifs obtenus par ces sauts sont les plus grands motifs non invalidés par un motif non fréquent déjà découvert, i.e. un motif de $\mathcal{B}d^-(F)$.

Cadre formel

Les résultats et définitions suivants sont issus des travaux préalablement effectués au sein de l'équipe pour l'algorithme *ZigZag* [MP03]. Soit $\mathcal{P}(R)$ l'espace de recherche des motifs fréquents et supposons qu'une partie E de cet espace ait été explorée. Soit $NF \subseteq E$

l'ensemble des motifs non fréquents de ce sous-ensemble. A partir de cet ensemble NF, on peut définir par anti-monotonie deux types de motifs dans $\mathcal{P}(R)$:

- ceux qui sont invalidés par anti-monotonie par au moins un motif non fréquent de NF.
- ceux qui restent valides, i.e. tous leurs sous-ensembles sont des motifs fréquents de E.

Soit $F_{opt}(NF)$ l'ensemble optimiste de motifs fréquents de $\mathcal{P}(R)$, défini comme suit :

$$F_{opt}(NF) \ = \ \{X \in \mathcal{P}(R) | \ \nexists Y \in NF, Y \subseteq X\}$$

La bordure positive de $F_{opt}(NF)$ est appelée *bordure positive optimiste* associée à NF. Soit X un motif de la bordure positive finale, alors X appartient à la bordure positive optimiste, où il existe un motif Y de la bordure positive optimiste tel que $X \subset Y$, i.e.

$$\forall X \in \mathcal{B}d^+(F), \exists Y \in \mathcal{B}d^+(F_{opt}(NF)) \text{ tel que } X \subseteq Y$$

A partir du théorème 2 section 3, la bordure positive optimiste associée à NF est donnée par :

$$\mathcal{B}d^+(F_{opt}(NF)) = \overline{Tr(F_{opt}(NF))}$$

avec $\overline{Tr(F_{opt}(NF))} = \{R \backslash Z \mid Z \in Tr(F_{opt}(NF))\}$

On a de plus la propriété suivante :

Propriété 9:

$$Si \ X \in \mathcal{B}d^+(F_{opt}(NF)) \ et \ X \in F \Rightarrow X \in \mathcal{B}d^+(F)$$

La bordure positive optimiste ne change pas si l'on restreint l'ensemble NF à ses motifs les plus petits, puisque les transversaux minimaux d'un hypergraphe sont les mêmes que ceux de l'hypergraphe simple associé. Par conséquent, chaque saut dans l'espace de recherche ou "dualisation", est un calcul de la bordure positive optimiste associée aux motifs non fréquents découverts et appartenant à la bordure négative.

D'après le théorème 2 section 3, lorsque $NF = \mathcal{B}d^-(F)$, on a $\mathcal{B}d^+(F) = \mathcal{B}d^+(F_{opt}(NF))$. Par conséquent, à chaque itération les nouveaux motifs non fréquents trouvés par l'algorithme vont permettre d'améliorer la bordure positive optimiste, et de se rapprocher de plus en plus de la bordure positive finale. Lorque la bordure négative est découverte, la dualisation génère la bordure positive finale, et l'algorithme s'arrête.

Justification de cette approche

Pour l'algorithme $ZigZag$ et l'extraction des DI les plus spécialisées, une propriété permettait de justifier l'approche optimiste [MP03]. A notre connaissance, les motifs fréquents n'ont pas une telle propriété. Toutefois, l'étude des jeux de données du FIMI [Goeb]

montre qu'expérimentalement, la probabilité qu'a un motif de taille k d'être fréquent sachant que tous ses sous-ensembles de taille $k-1$ sont fréquents, croît généralement avec k, comme le montre la figure 7.1 pour quelques jeux de données du FIMI.

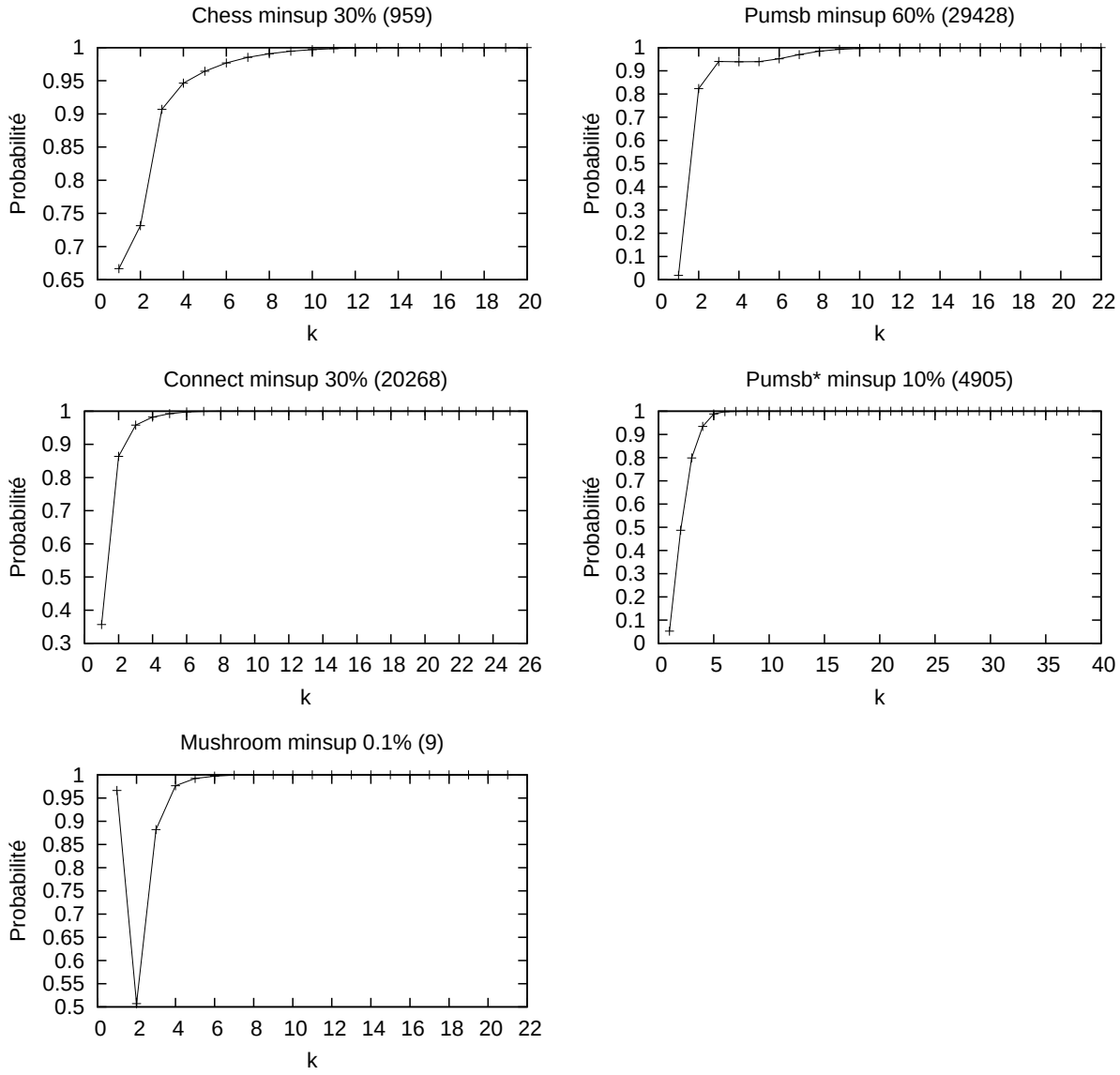

FIG. 7.1 – Probabilité d'un motif de taille k d'être fréquent sachant que tous ses sous-ensembles sont fréquents

Cette probabilité, notée $p(X \in F_k \mid \forall Y \subset X, Y \in F)$, a été obtenue à partir des résultats des expérimentations présentés dans la partie précédente. Cette mesure a été calculée de la manière suivante :

Définition 10:

$$p(X \in F_k \mid \forall Y \subset X, Y \in F) = \frac{|F_k|}{|F_k| + |\mathcal{B}d_k^-(F)|}$$

avec $|X| = k$, F_k *l'ensemble des motifs fréquents de taille* k, *et* $\mathcal{B}d_k^-(F)$ *l'ensemble des motifs non fréquents minimaux de taille* k.

La somme $|F_k| + |\mathcal{B}d_k^-(F)|$ correspond au nombre de motifs de taille k de l'espace de recherche ayant tous leurs sous-ensembles fréquents. Notons que c'est aussi le nombre de motifs candidats C_k générés à chaque itération par *Apriori* [AS94].

Comme le montre la figure 7.1, cette mesure augmente progressivement avec la taille des motifs. Cette remarque est vraie pour l'ensemble des jeux de données étudiés. Autrement dit, si tous les sous-ensembles de taille k d'un motif X sont fréquents, alors X a plus de chances d'être fréquent lorsque k augmente. Par conséquent, la proportion de motifs fréquents générés par un algorithme tel qu'*Apriori* aura tendance à devenir de plus en plus importante au cours des itérations. L'algorithme élaguera donc de moins en moins l'espace de recherche. Par exemple, pour *Connect* avec un seuil de 30% (figure 7.1), à partir du niveau 4, les motifs générés par *Apriori* seront en grande majorité fréquents (plus de 99%). *Apriori* ne fera donc que générer des motifs fréquents jusqu'au niveau 25 (taille du plus grand motif fréquent), ce qui se traduit pour cet exemple par plus d'un milliard de motifs à tester ! Ce résultat justifie ainsi l'intérêt de faire des sauts dans l'espace de recherche à partir d'un certain niveau.

7.1.3 De la bordure positive à la bordure négative

Après une approche par niveau, l'algorithme alterne des sauts ou dualisations entre les deux bordures en construction. Comme nous venons de le voir, les dualisations de la bordure négative à la bordure positive permettent de générer les plus grands motifs non invalidés par un motif de la bordure négative déjà découvert. Les motifs obtenus seront soit fréquents et appartiendront à la bordure positive finale, soit ils seront non fréquents et dans ce cas l'algorithme devra corriger l'erreur faite.

Pour corriger cette erreur, *ABS* s'appuie sur le théorème 1 section 3 et sur les dualisations effectuées dans *Dualize and Advance* (section 4.3) pour estimer la bordure négative en fonction de la bordure positive en cours. Intuitivement, les motifs candidats générés seront les plus petits motifs non inclus dans un motif de la bordure positive. Les nouveaux motifs de la bordure négative ainsi découverts seront utilisés pour corriger la bordure positive optimiste lors de la dualisation suivante.

Notons $\mathcal{B}d_i^+$ (resp. $\mathcal{B}d_i^-$) le sous-ensemble de la bordure positive $\mathcal{B}d^+(F)$ (resp. $\mathcal{B}d^-(F)$) découverts à l'itération i. On a $\forall i < j, \mathcal{B}d_i^+ \subseteq \mathcal{B}d_j^+$ et $\mathcal{B}d_i^- \subseteq \mathcal{B}d_j^-$. Les motifs de $\mathcal{B}d_i^+$ sont obtenus à partir des motifs de $\mathcal{B}d_i^-$, et les motifs de $\mathcal{B}d_{i+1}^-$ sont obtenus à partir des motifs $\mathcal{B}d_i^+$. Soit $C_{\leq\, i}$ l'ensemble des motifs candidats générés avant et pendant l'itération i.

A l'itération $i+1$, les motifs candidats générés pour corriger les motifs non fréquents de la bordure positive optimiste peuvent être obtenus à partir de la formule suivante.

Définition 11:

$$C_{i+1} = Tr(\overline{\mathcal{B}d_i^+}) - C_{\leq\ i}$$

De la même manière que pour la dualisation de la bordure négative vers la bordure positive optimiste, les motifs non fréquents générés par cette dualisation appartiennent à la bordure négative finale [GKM+03], i.e.

Propriété 10:

$$Si\ X \in C_{i+1}\ et\ X \notin F \Rightarrow X \in \mathcal{B}d^-(F)$$

La différence majeure avec une méthode par niveau ou en profondeur est que les motifs candidats générés par cette dualisation peuvent être de taille supérieure à $i+1$, où i est la taille des motifs de l'itération précédente. L'exemple suivant illustre cela.

Exemple 4:
Soit r la base de données de transactions sur $R = \{A, B, C, D, E, F, G, H, I\}$ suivante :

$$r = \{ABCHI, ABDGI, ABGHI,\quad ACDFI, ACFHI, ADFGI, AFGHI, BCDEI,$$
$$BCEHI, BDEGI, BEGHI,\quad CDEFI, CEFHI, DEFGI, EFGHI, ABCD\}$$

Pour un seuil minimum de support absolu de un, les bordures des motifs fréquents de r sont égales à

$$\mathcal{B}d^-(F) = \{AE, BF, CG, DH, ABCDI\}$$
$$\mathcal{B}d^+(F) = \{ABCHI, BCEHI, ABDGI,\quad BDEGI, ABGHI, BEGHI, ACDFI, CDEFI,$$
$$ACFHI, CEFHI, ADFGI,\quad DEFGI, AFGHI, EFGHI, BCDEI, ABCD\}$$

Après un parcours par niveau jusqu'au niveau deux, quatre motifs non fréquents minimaux de taille deux ont été découverts, i.e. $\mathcal{B}d^-_{\leq\ 2}(F) = \{AE, BF, CG, DH\}$. Supposons que l'algorithme stoppe son approche par niveau. Alors, ces motifs non fréquents sont utilisés pour la première dualisation. La bordure positive optimiste obtenue est $\{ABCHI, ABDGI, ABGHI, ACDFI, ACFHI, ADFGI, AFGHI, BCDEI, BCEHI,$ $BDEGI, BEGHI, CDEFI, CEFHI, DEFGI, EFGHI\}$. La dualisation de la bordure positive en construction à la bordure négative génère le motif candidat $ABCD$ de taille 4, et aucun motif de taille 3 n'est construit.

Suite à cet exemple, nous suspectons l'algorithme $Pincer - Search$ [LK98] de ne pas être complet. Leur stratégie est proche de la nôtre tout en étant entièrement empirique,

notamment lors de la génération des motifs candidats. Les motifs candidats de $Pincer - Search$ correspondent uniquement à ceux de taille $i+1$ dans C_{i+1} pour ABS. Puisque cet algorithme arrête son exploration lorsqu'il n'y a plus de motifs candidats générés, certains motifs peuvent être manqués. Sur l'exemple 4, après le niveau 2 aucun motif candidat n'est généré, et par conséquent le motif fréquent maximal $ABCD$ semble ne jamais être trouvé par $Pincer - Search$.

7.1.4 Les aspects adaptatifs

Fin de l'approche par niveau et première dualisation

L'un des aspects adaptatifs les plus importants est de déterminer à quel niveau l'approche par niveau doit être arrêtée, et donc à quel moment les sauts doivent débuter. L'objectif est d'estimer s'il est intéressant de dualiser la bordure négative en cours pour obtenir la bordure positive optimiste.

Nous avons identifié quatre paramètres spécifiques d'une itération de l'approche par niveau, et qui peuvent être obtenus dynamiquement sans surcoût :
- le niveau actuel i ;
- $|\mathcal{B}d^-_{\leq i}(F)|$, le nombre de motifs de la bordure négative de taille inférieure ou égale à i et qui ont déjà été découverts ;
- $|F_i|$, le nombre de motifs fréquents découverts au niveau i ;
- $|C_i|$, le nombre de motifs candidats générés au niveau i.

Tout d'abord, aucun saut n'est autorisé avant un certain niveau. En effet, l'approche par niveau est très efficace en pratique pour explorer les premiers niveaux de l'espace de recherche. Dans nos expérimentations, dualiser avant le niveau quatre par exemple n'apporte aucune amélioration.

Comparons maintenant $|F_i|$ et $|C_i|$. Par construction, les motifs candidats générés par *Apriori* sont soit des motifs fréquents, soit des motifs non fréquents minimaux, i.e. $C_i = F_i \cup \mathcal{B}d^-_i(F)$. Le ratio $|F_i|/|C_i|$ représente donc la proportion de motifs fréquents générés, ou comme nous l'avons introduit précédemment la probabilité qu'a un motif de taille i d'être fréquent sachant que tous ses sous-ensembles le sont. Or comme nous l'avons constaté, cette probabilité tend à être strictement croissante dans une grande partie des cas. Supposons qu'à un niveau donné cette probabilité soit proche de un. Par conséquent, l'espace de recherche restant à étudier sera principalement composé de motifs fréquents. Ce type de configuration n'est pas favorable à une approche par niveau telle qu'*Apriori*, dont l'efficacité réside principalement dans l'exploitation des motifs non fréquents minimaux pour élaguer l'espace de recherche. A l'opposé, les stratégies de sauts sont beaucoup plus efficaces. Les expérimentations présentées dans le chapitre 6 ont montré cela lorsque la distance entre les deux bordures était importante. Dans ce cas il est possible, en utilisant la propriété d'anti-monotonie du prédicat, de déduire la bordure positive des motifs

fréquents. Lorsque $|F_i|/|C_i|$ est proche de un, il est donc préférable de stopper l'approche par niveau et d'effectuer la phase de dualisation.

Les grands motifs "presque fréquents"

Considérons maintenant les motifs obtenus par une dualisation à partir de la bordure négative en construction. Soit X un de ces motifs. Deux cas sont possibles : X est fréquent ou X est non fréquent. Dans ce dernier cas, nous proposons d'estimer l'erreur faite par le saut afin de choisir la stratégie la plus adaptée pour la corriger.

En effet, si un motif est non fréquent mais "proche" de la bordure positive finale, un parcours de l'espace de recherche des motifs les plus grands aux plus petits sera particulièrement adapté. Dans le cas contraire, si le motif est "éloigné" de la bordure positive, il sera plus intéressant d'étudier les motifs "par le bas", i.e des plus petits aux plus grands, afin de découvrir de nouveaux motifs de la bordure négative et de corriger le saut.

Pour estimer la distance entre un motif non fréquent et la bordure positive, l'algorithme utilise la mesure d'erreur suivante. Soit δ un seuil fixé par l'utilisateur et un seuil minimum de support $minsup$, une mesure d'erreur pour un motif non fréquent X, notée $error(X, minsup)$, peut être définie de la manière suivante :

Définition 12:

$$error(X, minsup) = 1 - \frac{support(r, X)}{minsup}$$

Cette mesure reflète la différence entre le support de X et le seuil minimum de support. Plus le support d'un motif est proche du seuil minimum de support, plus la mesure d'erreur tend vers zéro.

A partir de cette mesure, deux sous-cas doivent être considérés pour les motifs non fréquents découverts par dualisation :
- $error(X, minsup) \leq \delta$: le motif X généré par dualisation est faux, mais des solutions doivent exister dans les sous-ensembles proches de X ;
- $error(X, minsup) > \delta$: le saut a été trop optimiste, aucune solution n'existe dans les sous-ensembles proches.

De plus, nous avons la propriété suivante.

Propriété 11:

$$X \subset Y \Rightarrow error(X, minsup) \leq error(Y, minsup)$$

Cette mesure est donc décroissante, mais pas strictement décroissante car le support peut rester constant lorsque X appartient à une classe d'équivalence contenant plusieurs motifs.

Notons $\mathcal{N}earBd_i^+$ l'ensemble des motifs "presque fréquents" découverts à l'itération i :

Définition 13:

$$\mathcal{N}earBd_i^+ = \{X \in \mathcal{B}d_i^+ \ / \ X \notin F \ et \ error(X, minsup) \leq \delta\}$$

7.2 L'algorithme

L'algorithme ABS permet de découvrir la bordure positive et la bordure négative des motifs fréquents. Il prend uniquement en paramètre d'entrée la base de données de transactions et le seuil minimum de support, les paramètres adaptatifs ont été fixés expérimentalement.

L'algorithme commence par explorer l'espace de recherche par niveau en utilisant *Apriori* (lignes 1-7 et 20-25). A chaque itération, la fonction $IsDualisationRelevant$ (ligne 8) étudie si l'approche par niveau doit être stoppée. Si c'est le cas, l'algorithme alterne des dualisations entre les deux bordures (lignes 8-19). Les fonctions $GenPosBorder$ et $GenNegBorder$ calculent chacune une des deux dualisations. $GenPosBorder$ permet de générer la bordure positive optimiste, et $GenNegBorder$ les motifs candidats utilisés pour corriger les motifs non fréquents très éloignés de la bordure positive. Ces calculs sont faits de manière incrémentale. Ils utilisent donc uniquement les motifs trouvés dans l'itération en cours.

Notons qu'à la ligne 12, $GenNegBorder$ prend en paramètre les motifs de la bordure positive et les motifs "presque fréquents" découverts dans la bordure positive optimiste. Les motifs "presques fréquents" sont ainsi considérés comme des motifs de la bordure positive, et sont donc utilisés pour élaguer l'espace de recherche. Ces motifs seront traités à la fin, ligne 18, par la méthode $ProcessNearPosBorder$ qui effectuera un parcours par niveau de "haut en bas" de l'espace de recherche, i.e. des motifs les plus grands aux plus petits, afin de découvrir les motifs fréquents maximaux manquants.

Soient n' la taille du plus grand motif de la bordure négative, et k le dernier niveau exploré par *Apriori*. L'algorithme fera donc dans le pire cas $2 \times (n' - k) + 1$ dualisations avant de découvrir les bordures. En effet, l'algorithme fera $n' - k + 1$ dualisations de la bordure négative vers la bordure positive, et $n' - k$ de la bordure positive vers la bordure négative.

Algorithme 1 *ABS* : Adaptive Borders Search

Entrée: une base de données de transactions r , un entier $minsup$
Sortie: $\mathcal{B}d^+(F)$ and $\mathcal{B}d^-(F)$

1: $F_1 = \{A \in r \mid support(r, A) \geq minsup\}$
2: C_2 =AprioriGen(F_1)
3: $i = 2$; $\mathcal{B}d_1^- = R - F_1$; $\mathcal{B}d_0^+ = \emptyset$; $\mathcal{N}earBd_0^+ = \emptyset$
4: **while** $C_i \neq \emptyset$ **do**
5: $F_i = \{X \in C_i \mid support(r, X) \geq minsup\}$
6: $\mathcal{B}d_i^- = \mathcal{B}d_{i-1}^- \cup (C_i - F_i)$
7: $\mathcal{B}d_{i-1}^+ = \mathcal{B}d_{i-2}^+ \cup \{X \in F_{i-1} \mid \forall Y \in F_i, X \not\subseteq Y\}$
8: **if** IsDualizationRelevant(i,$|\mathcal{B}d_i^-|$,$|F_i|$,$|C_i|$)=TRUE **then**
9: $\mathcal{B}d_i^+ = \{X \in GenPosBorder(\mathcal{B}d_i^-) \mid support(r, X) \geq minsup\}$
10: $\mathcal{N}earBd_i^+ = \{X \in GenPosBorder(\mathcal{B}d_i^-) \mid error(X, minsup) \leq \delta\}$
11: **while** $\mathcal{B}d_i^+ \neq \mathcal{B}d_{i-1}^+$ **do**
12: $\mathcal{B}d_i^- = \{X \in GenNegBorder(\mathcal{B}d_i^+ \cup \mathcal{N}earBd_i^+) \mid support(r, X) < minsup\}$
13: $\mathcal{B}d_{i+1}^+ = \{X \in GenPosBorder(\mathcal{B}d_i^-) \mid support(r, X) \geq minsup\}$
14: $\mathcal{N}earBd_{i+1}^+ = \{X \in GenPosBorder(\mathcal{B}d_i^-) \mid error(X, minsup) \leq \delta\}$
15: $\mathcal{N}earBd_{i+1}^+ = \mathcal{N}earBd_i^+ \cup \mathcal{N}earBd_{i+1}^+$
16: $i = i + 1$
17: **end while**
18: ProcessNearPosBorder($\mathcal{N}earBd_{i+1}^+$, $\mathcal{B}d_i^+$, $\mathcal{B}d_{i-1}^-$)
19: **Return** $\mathcal{B}d_i^+$ and $\mathcal{B}d_{i-1}^-$ and **exit**
20: **end if**
21: C_{i+1} =AprioriGen(F_i)
22: $i = i + 1$
23: **end while**
24: $\mathcal{B}d_{i-1}^+ = \mathcal{B}d_{i-2}^+ \cup F_{i-1}$
25: **Return** $\mathcal{B}d_{i-1}^+$ and $\mathcal{B}d_{i-1}^-$

Rappelons qu'en comparaison l'algorithme *Dualize and advance* (section 4.3) fait $|\mathcal{B}d^+(F)|$ dualisations.

Calcul des transversaux minimaux

Le calcul des transversaux minimaux est une étape importante de notre algorithme, et conditionnne en partie les performances de l'implémentation. Dans notre cas, les transversaux minimaux étant calculés à partir de l'hypergraphe constitué des motifs des bordures en construction, une approche incrémentale est préférable.

Ce calcul peut, pour notre problème, être amélioré en prenant en compte plusieurs facteurs :

– Tout d'abord, les motifs de la bordure positive optimiste obtenus en complémentant les transversaux minimaux ne doivent pas être de taille inférieure ou égale à k, où

k est le niveau actuel "par le bas". En effet, tous les motifs fréquents maximaux jusqu'à ce niveau ont déjà été trouvés par *Apriori*.
- Ensuite, tout transversal minimum obtenu à l'étape i et qui a donné un motif fréquent dans la bordure positive optimiste, restera transversal minimum quelles que soient les nouvelles arêtes de l'hypergraphe.

Plus formellement, soit $\mathcal{B}d^-(F) = \{A_1, ..., A_i, A_{i+1}, ..., A_m\}$. Notons $Tr_i(\mathcal{B}d^-(F))$ l'ensemble de transversaux minimaux obtenus par une méthode incrémentale après ajout des i premiers motifs de $\mathcal{B}d^-(F)$. Soient $S_i = \{X \in Tr_i(\mathcal{B}d^-(F)) \mid X \cap A_{i+1} \neq \emptyset\}$ et $B_i = \{X \in Tr_i(\mathcal{B}d^-(F)) \mid X \cap A_{i+1} = \emptyset\}$. Un grand nombre de méthodes incrémentales utilise la caractérisation générique suivante pour exprimer $Tr_i(\mathcal{B}d^-(F))$ en fonction de $Tr_{i+1}(\mathcal{B}d^-(F))$:

$$Tr_{i+1}(\mathcal{B}d^-(F)) = S_i \cup \{X \cup \{a\} \mid X \in B_i, a \in A_{i+1} \text{ tel que } \mathbf{Cond} \}$$

avec **Cond** des conditions que doivent respecter X et a, pour que $X \cup \{a\}$ soit un transversal minimum du nouvel hypergraphe. Ces conditions dépendent de la méthode utilisée.

A partir des remarques précédentes et de ces définitions, on obtient les propriétés suivantes.

Propriété 12:

$$Tr_{i+1}(\mathcal{B}d^-(F)) = S_i \cup \{X \cup \{a\} \mid X \in B_i, a \in A_{i+1} \textit{ tel que } \textit{Cond et } \mathbf{|X| \leq n\text{-}k\text{-}1} \ \} \\ \Leftrightarrow \overline{Tr_m(\mathcal{B}d^-(F))} = \{Z \in \mathcal{B}d^+(F) \mid |Z| \geq k\}$$

avec $m = |\mathcal{B}d^-(F)|$ *et* $i < m$.

Autrement dit, à chaque calcul de $Tr_i(\mathcal{B}d^-(F))$, seuls les transversaux minimaux dont la taille est inférieure ou égale à $n - k$ sont conservés, les autres peuvent être directement supprimés.

Propriété 13:

$$X \in Tr_i(\mathcal{B}d^-(F)) \textit{ et } \overline{X} \in \mathcal{B}d^+(F) \Leftrightarrow X \in Tr_m(\mathcal{B}d^-(F))$$

avec $m = |\mathcal{B}d^-(F)|$ *et* $i \leq m$.

Pour calculer $Tr_{i+1}(\mathcal{B}d^-(F))$, ce type de méthode parcourt l'ensemble des transversaux minimaux de $Tr_i(\mathcal{B}d^-(F))$ afin de tester si chaque ensemble reste un transversal minimum après l'ajout de l'arête A_{i+1}. Dans notre contexte, il n'est donc pas nécessaire de tous les tester car on sait déjà que les transversaux minimaux dont le complémentaire appartient à $\mathcal{B}d^+(F)$ resteront transversaux minimaux après l'ajout de tous les motifs de $\mathcal{B}d^-(F)$.

7.3 Expérimentations

7.3.1 Implémentation

L'algorithme a été développé en C++ en utilisant la STL. L'implémentation de l'algorithme *Apriori* de C. Borgelt [Bor03] a été choisie pour effectuer le parcours par niveau car elle est l'implémentation d'*Apriori* la plus performante actuellement [BZ03]. Elle a été modifiée pour stocker la bordure négative et s'arrêter au bout d'un certain nombre d'itérations en fonction de la stratégie adaptative définie précédemment. Pour le calcul incrémental des transversaux minimaux, l'approche de Demetrovics et Thi [DT95] a été utilisée. L'implémentation d'*ABS* utilise des arbres préfixés ou *trie* pour stocker les transactions et les motifs.

7.3.2 Expérimentations

Les expérimentations ont été réalisées en deux temps. Dans un premier temps, nous avons réalisé une étude expérimentale de l'algorithme à partir des jeux de données disponibles sur le site du FIMI [Goeb]. Nous avons étudié l'influence du niveau d'arrêt de l'approche par niveau sur les performances d'*ABS*. Puis l'impact des motifs "presque fréquents" est étudié pour différents seuils de mesure d'erreur. Nous avons finalement comparé *ABS* avec quatre algorithmes représentatifs : *Apriori* et *Eclat* [Bor03], $FPmax*$ [GZ03] et IBE [SU03].

Notons que l'algorithme IBE est dérivé de l'algorithme *Dualize and Advance* [GKM+03] présenté en section 4.3, et exploite donc lui aussi la notion de dualisation pour effectuer des sauts dans l'espace de recherche.

Dans un deuxième temps, l'implémentation d'*ABS* a participé au banc d'essais de FIMI'04 [BGZ04]. Ces expérimentations ont permis de comparer les performances d'*ABS* avec un plus grand nombre d'algorithmes de découverte des motifs fréquents maximaux, dont les plus performants actuellement, et sur davantage de jeux de données.

Etude expérimentale des aspects adaptatifs et premières expérimentations

Cette première partie des expérimentations a été réalisée sur un pentium 4.3Ghz avec 1Go de mémoire. Le système d'exploitation est Linux Redhat 7.3, et le compilateur est gcc 3.96.

Dans la figure 7.2, nous avons forcé l'arrêt de l'approche par niveau, et le début de la phase de dualisation, à différents niveaux (de 3 à 8) sur les jeux de données *Connect* et $Pumsb*$ pour des seuils minimums de support respectivement de 60% et 30%. Comme le montrent les résultats obtenus pour ces jeux de données, le niveau auquel a lieu le change-

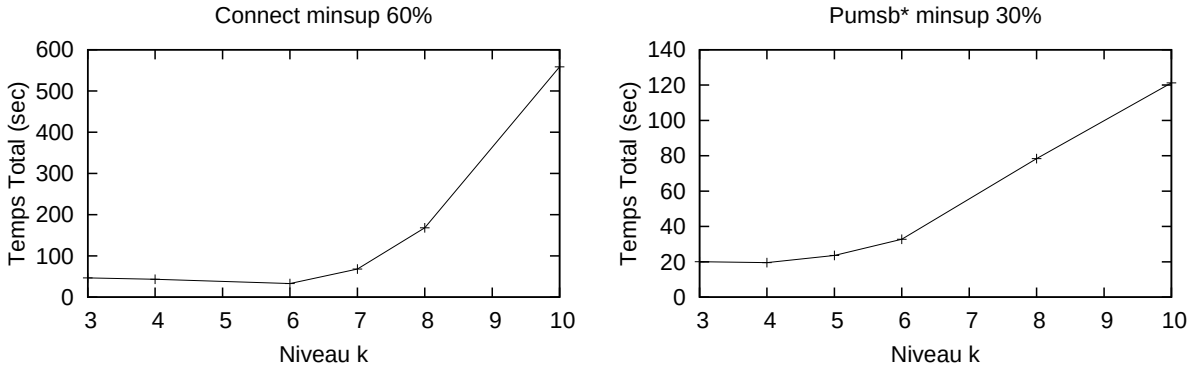

FIG. 7.2 – Première dualisation au niveau k pour connect (gauche) et pumsb* (droite)

ment de stratégie a une forte influence sur les performances de l'algorithme. Par exemple, sur Connect (figure 7.2 de gauche) les temps d'exécution peuvent varier jusqu'à un facteur 20 selon le niveau choisi pour changer de stratégie. Ces résultats confirment donc la nécessité de fixer dynamiquement ce paramètre, et justifient ainsi une approche adaptative. De plus, pour tous les jeux de données testés, notre fonction $IsDualizationRelevant$ a toujours trouvé dynamiquement le meilleur niveau pour arrêter l'approche par niveau et commencer les dualisations.

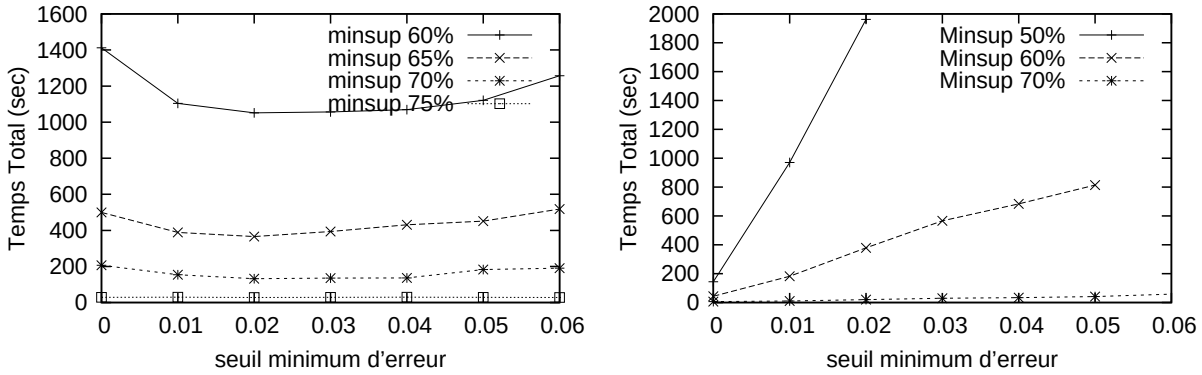

FIG. 7.3 – Temps d'exécution d'ABS sur $Pumsb$ (gauche) et $Connect$ (droite) pour différents seuils de mesure d'erreur

L'influence de la mesure d'erreur est évaluée sur la figure 7.3. Pour rappel, un motif de la bordure positive optimiste est "presque fréquent" lorsque son erreur est inférieure à un seuil δ. Nous allons donc ici étudier l'influence de ce seuil sur les performances. Pour le jeu de données $Pumsb$, la prise en compte des motifs "presque fréquents" s'avère intéressante lorsque le seuil est proche de 0.002. Néanmoins, pour $Connect$, la prise en compte des motifs "presque fréquents" n'apporte aucune amélioration. L'existence de grandes classes d'équivalence pour ce jeu de données (section 6.2.2) implique que le support n'est pas strictement décroissant. Par conséquent, les motifs identifiés comme "presque fréquents" peuvent s'avérer loin de la bordure positive. Sur ce type de jeu de données, l'exploration par niveau de "haut en bas" de l'espace de recherche est vouée à l'échec.

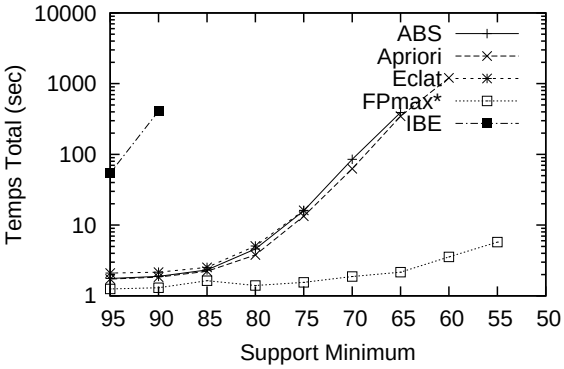

FIG. 7.4 – Temps d'exécution d'*ABS* sur *Pumsb*

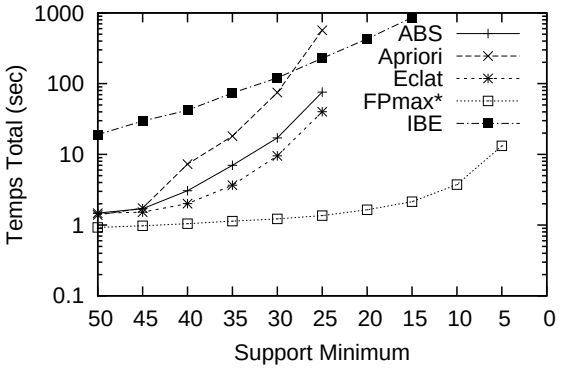

FIG. 7.5 – Temps d'exécution d'*ABS* sur *Pumsb∗*

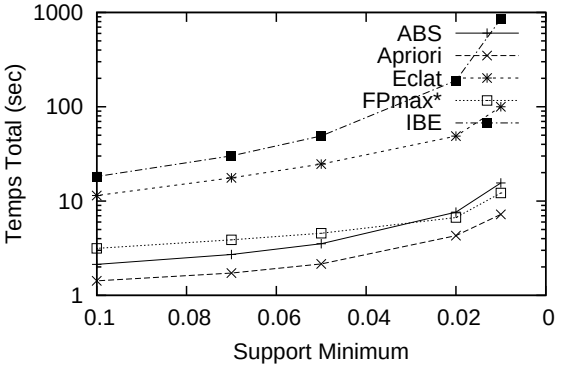

FIG. 7.6 – Temps d'exécution d'*ABS* sur *Retail*

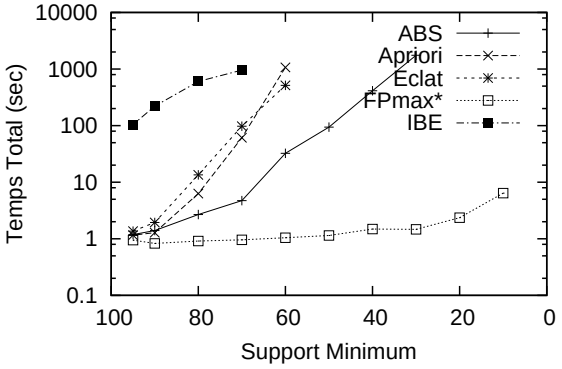

FIG. 7.7 – Temps d'exécution d'*ABS* sur *Connect*

D'après les figures 7.4, 7.6 et 7.7, *ABS* est loin d'égaler les meilleures implémentations connues pour découvrir les motifs fréquents maximaux tels que *FPmax** [GZ03]. Toutefois, il tend à être meilleur que *IBE* [SU03] sur la majeure partie de nos expérimentations. *IBE* est la seule implémentation disponible sur le site du FIMI s'appuyant sur le concept de dualisation. Cette différence est en partie due au nombre de dualisations faites par *IBE*, qui est dans la taille de la bordure positive, alors que le nombre de dualisations réalisées par *ABS* est dans la taille du plus grand motif de la bordure négative. En effet, *IBE* fait une dualisation pour chaque motif de la bordure positive, et cette dualisation lui permet de trouver un seul nouveau motif de la bordure positive. Dans le pire cas, *ABS* fait deux dualisations pour chaque niveau à partir du dernier exploré par *Apriori* jusqu'au niveau contenant le plus grand motif de la bordure négative, en supposant que les motifs candidats générés sont de taille $i+1$ (sinon l'algorithme fait encore moins de dualisations). De plus, à chaque itération, *ABS* peut trouver plusieurs motifs de la bordure positive, et donc élaguer de plus grandes parties de l'espace de recherche.

Dans la figure 7.5, *IBE* a de meilleures performances qu'*ABS* pour des seuils minimums de support petits (moins de 20%). Pour ces seuils de support, la taille de la bordure positive reste petite (moins de 5000 motifs), alors que celle de la bordure négative dépasse 10^6 motifs, avec certains motifs de grande taille. C'est le pire cas pour *ABS*.

Dans la figure 7.6, *ABS* se comporte comme prévu de la même manière qu'*Apriori*. En effet, la bordure positive de *Retail* est composée de "petits" motifs, et *Apriori* est le meilleur algorithme pour ce type de jeux de données.

Pour des jeux tels que *Connect* (figure 7.7), *ABS* n'est pas aussi efficace que les meilleures implémentations, par exemple *FPmax**, mais améliore *Apriori* par un facteur de dix, et est meilleur que *Eclat* et *IBE*.

Participation au banc d'essai de FIMI

L'implémentation d'*ABS* a par la suite participé au banc d'essai de FIMI'04 [BGZ04]. Les résultats obtenus pendant ces tests confirment ceux obtenus précédemment. Comme le montre la figure 7.8 pour différents types de jeux de données, *ABS* est plus particulièrement efficace pour des jeux de données tels que *retail*, *Pumsb**, *Connect* ou *Mushroom*. Pour rappel, ces jeux de données ont des motifs fréquents maximaux de petite taille, ou des motifs fréquents maximaux de grande taille mais avec une bordure négative composée de motifs relativement petits (i.e. une grande distance entre les bordures, chapitre 6).

Pour des jeux de données tels que *Chess* ou *Pumsb*, les motifs de la bordure négative sont grands et leur nombre peut être important. Par conséquent, selon le niveau où sera faite la première dualisation, il y aura soit une grande partie de l'espace de recherche explorée par niveau, soit un grand nombre de dualisations avec beaucoup de motifs à prendre en compte à chaque fois. Les performances d'*ABS* pour ces jeux de données sont donc moins bonnes (figure 7.8).

Toutefois, la stratégie adaptative permet dans la majeure partie des cas d'utiliser

l'approche la plus adaptée au jeu de données. Par exemple, pour *Retail*, l'approche par niveau est tout le temps utilisée car les motifs fréquents maximaux sont petits. Ceci se traduit sur la figure 7.8 par des performances identiques à *Apriori*. Dans ce cas, *ABS* est parmi les algorithmes les plus performants. A l'opposé, pour des jeux de données tels que *Connect*, l'approche par niveau est stoppée rapidement au profit des dualisations, ce qui permet de beaucoup améliorer les performances.

La stratégie d'*ABS* change en fonction du type de jeux de données étudiés, mais aussi en fonction du seuil minimum de support. En effet, comme le montre la figure 7.8 pour *Mushroom*, l'algorithme utilise l'approche par niveau jusqu'à un seuil minimum de support de 500 environ, car jusqu'à ce seuil les motifs fréquents maximaux restent relativement petits. Pour des seuils minimums de support plus petits, les motifs fréquents maximaux deviennent plus longs, l'algorithme change donc sa stratégie, ce qui se traduit sur la figure 7.8 par des performances meilleures qu'*Apriori* .

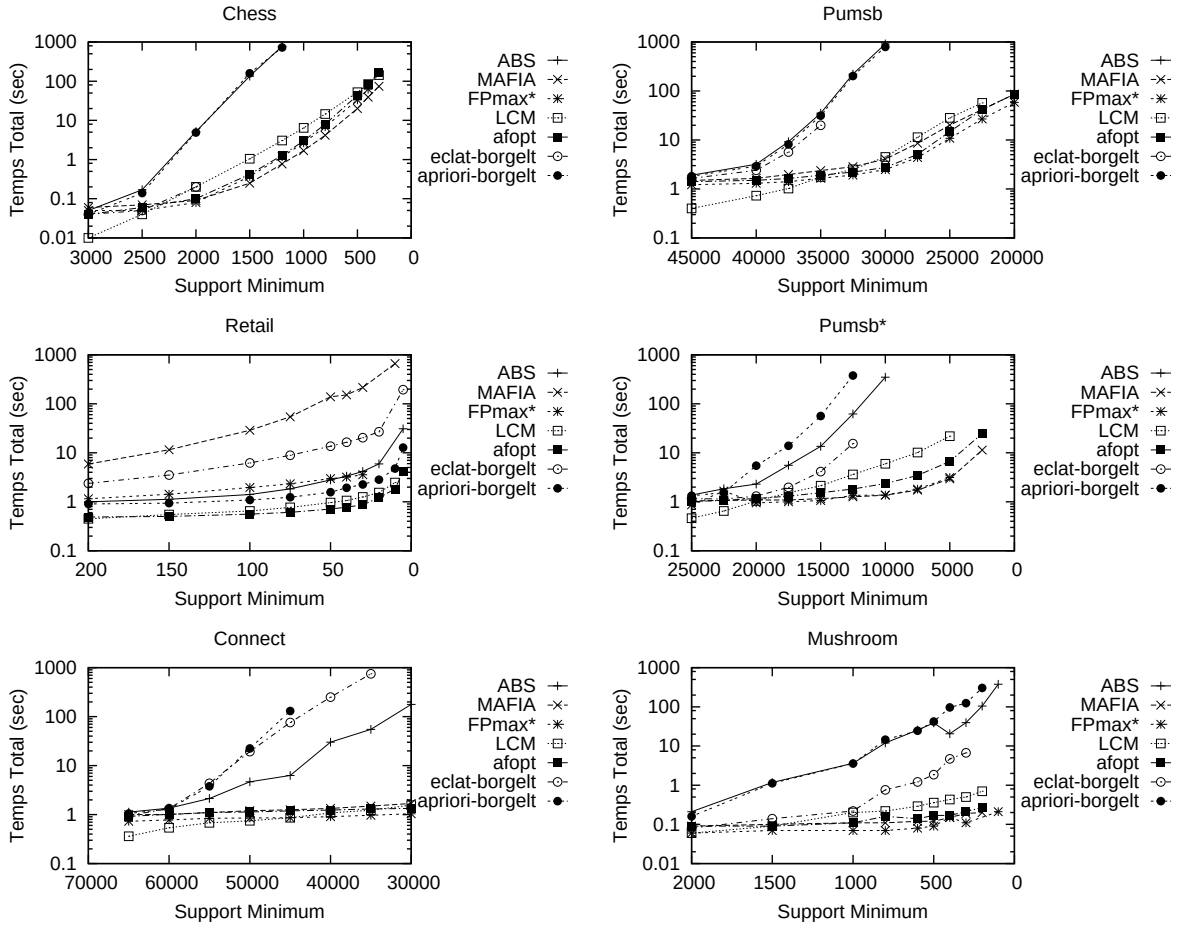

FIG. 7.8 – Performances d'*ABS* à FIMI'04

7.4 Discussion

Les expérimentations réalisées ont mis en évidence l'intérêt de cette approche pour traiter deux types de jeux de données :

- des jeux de données ayant une bordure positive composée de "petits" motifs ;
- des jeux de données ayant une grande différence de taille entre les motifs des bordures positive et négative.

Si on se réfère à la classification faite en section 6.3 par rapport à la "distance" entre les deux bordures, ces jeux de données sont de type II et de type III. Par ailleurs, à partir de ces résultats, il est apparu que le meilleur niveau pour effectuer la première dualisation, et trouvé dynamiquement par l'algorithme, correspond approximativement au niveau où le nombre de motifs découverts de la bordure négative décroît. La figure 7.9 illustre cela pour les jeux de données *Connect* et $Pumsb*$ pour des seuils minimums de support fixés. En effet, pour *Connect* le meilleur niveau pour stopper *Apriori* et effectuer la première dualisation est 6, ce qui correspond au niveau où le nombre de motifs découverts de la bordure négative diminue. La même chose peut être observée pour $Pumsb*$ (en faisant abstraction du nombre de motifs de bordure négative de taille 1).

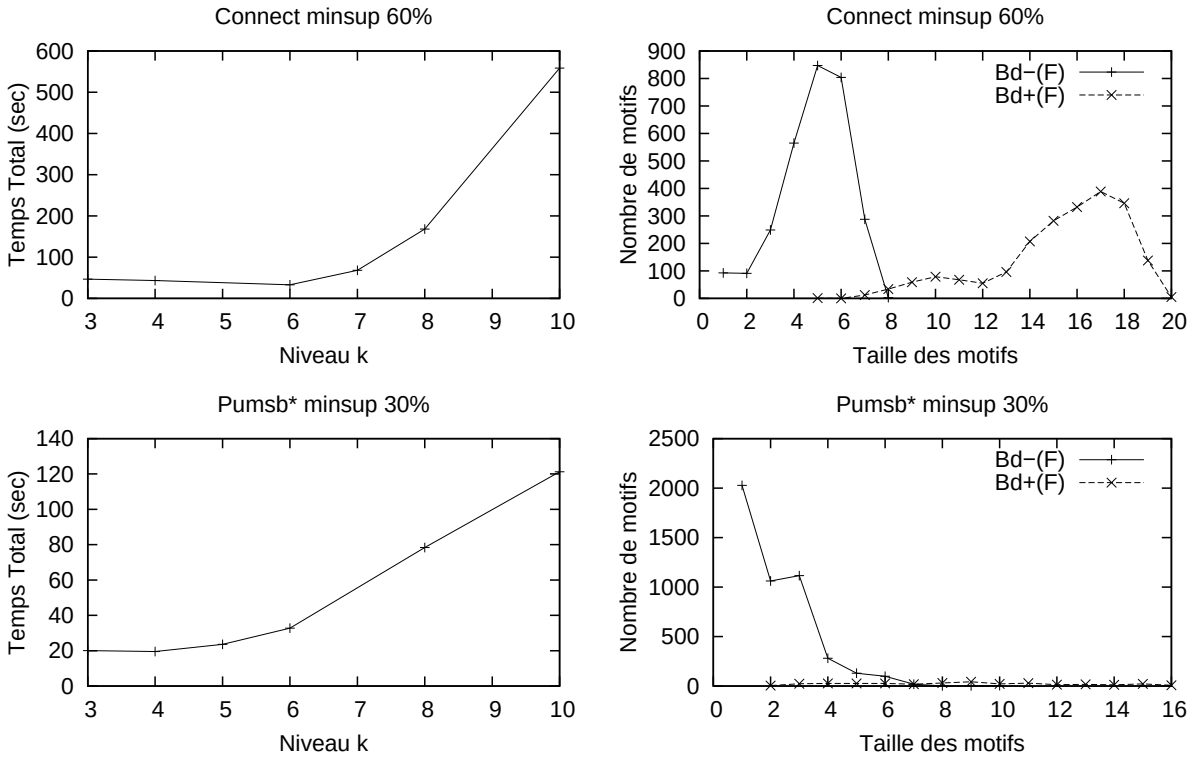

FIG. 7.9 – Niveau k de la première dualisation et distribution des bordures

De manière générale, il est aussi apparu qu'ABS était moins performant que les meilleurs algorithmes tels que $FPmax*$ [GZ03] ou LCM [UKA04]. Deux facteurs ex-

pliquent ce résultat mitigé : 1) le comptage du support n'est pas optimisé dans notre algorithme et 2)le coût des dualisations reste élevé pour des hypergraphes de grande taille.

Tout d'abord, il est important de noter que pour les problèmes de découverte des motifs fréquents, les performances des algorithmes sont fortement liées aux optimisations mises en place pour calculer le support. Pour constater cela, il suffit d'observer les performances de l'implémentation de $k-DCI$ [OLP+03] par rapport à celles d'*Apriori* [Bor03] obtenues pendant les expérimentations de FIMI [Goeb]. $k-DCI$ applique la même stratégie qu'*Apriori* en optimisant en plus le comptage du support. Pourtant, les performances de $k-DCI$ sont toujours bien meilleures que celles d'*Apriori*. Dans notre cas, l'objectif n'était pas d'optimiser le comptage du support mais de développer une stratégie adaptative et générique d'exploration de l'espace de recherche. Par conséquent, nous n'avons pas optimisé cette opération, ce qui explique la différence de performance avec les autres implémentations.

De plus, les dualisations faites par ABS reviennent à calculer de manière incrémentale les transversaux minimaux d'un hypergraphe composé des motifs de la bordure négative. Or comme nous l'avons vu, cette opération peut être très coûteuse. Toutefois, comme cela a été précisé dans [GKM+03], le calcul des transversaux n'accède pas aux données. Par conséquent, si la quantité de données est très importante, un calcul de transversaux minimaux pourra être moins coûteux que simplement lire les données.

Pour conclure, cet algorithme est à notre connaissance la première approche à adapter *dynamiquement* sa stratégie de parcours de l'espace de recherche. Cette stratégie adaptative consiste dans un premier temps à déterminer dynamiquement à quel moment l'algorithme doit passer d'une exploration par niveau, à une approche effectuant des "sauts" dans l'espace de recherche. Dans un second temps, la stratégie adaptative permet d'exploiter les motifs "presque vrais" découverts pendant ces sauts pour explorer différemment certaines parties de l'espace de recherche. Les expérimentations ont montré l'intérêt de ce type de méthode, même si les résultats expérimentaux sont mitigés, dus à l'importance des techniques de comptage pour ce problème. Par ailleurs, l'efficacité d'ABS pourrait être grandement améliorée en appliquant en plus une stratégie adaptative de comptage du support telle que celle utilisée dans $k-DCI$ [OLP+03].

Un autre intérêt de cette stratégie adaptative est de pouvoir être appliquée en théorie à tout problème d'extraction de motifs intéressants conforme au cadre théorique présenté dans le chapitre 3. Plus généralement, ceci soulève le problème de la réutilisation des algorithmes d'extraction de motifs fréquents pour résoudre tout autre problème équivalent. Cet aspect va être développé dans la partie suivante.

Quatrième partie

Vers des solutions génériques pour l'extraction de motifs intéressants

Chapitre 8

Contexte et état de l'art

Sommaire

Un grand nombre de problèmes ont été identifiés comme "équivalents" à un problème d'extraction de motifs fréquents. Le cadre théorique présenté dans le chapitre 3 les formalise sous la forme de problèmes "d'extraction de motifs intéressants" (motif au sens large).

Cette notion d'équivalence reste néanmoins théorique, leur résolution à partir des travaux réalisés pour l'extraction des motifs fréquents soulève plusieurs questions :

- Peut-on appliquer les algorithmes d'extraction de motifs fréquents à d'autres problèmes "équivalents", si oui à quel prix ?
- Peut-on utiliser ou adapter les solutions logicielles existantes pour les motifs fréquents dans le contexte générique de l'extraction de motifs intéressants ?

Notre équipe a notamment été confrontée à ces problèmes lors de l'utilisation d'algorithmes d'extraction de motifs fréquents pour résoudre des problèmes totalement différents tels que l'extraction des DF [LPL00], des DI [MLP02, MP03] et la résolution de certaines phases du processus de réécriture de requêtes dans un système d'intégration de sources de données hétérogènes [JPR+05, JF06].

Dans la suite de ce chapitre, nous allons présenter des problèmes de découverte de connaissances rentrant dans le cadre théorique considéré et étudier comment se situent

les solutions existantes par rapport aux deux questions abordées ici.

8.1 Une grande variété d'applications

Dans cette section, nous donnons quelques exemples de problèmes d'extraction de motifs intéressants entrant dans le cadre présenté au chapitre 3 et qui ont été étudiés par notre équipe.

8.1.1 Les motifs fréquents

Soit r une base de données de transactions sur R . Le problème de découvrir l'ensemble des motifs fréquents dans r par rapport à un seuil minimum de support donné peut être reformulé dans le cadre précédent.

Le langage $\mathcal{L}$ est composé de l'ensemble des parties de R , i.e. $\mathcal{P}(R)$. La base de données considérée est la base de données de transactions. Le prédicat "être fréquent", noté $freq$, est défini de la façon suivante : Soit $X \in \mathcal{L}, freq(r, X, minsup)$ est vrai si $support(r, X) \geq minsup$. La relation d'ordre est l'inclusion $\subseteq$. La thérorie $Th(r, \mathcal{P}(R), freq)$ est l'ensemble des motifs fréquents.

Le support étant décroissant par rapport à l'inclusion, ce prédicat est anti-monotone. Par conséquent, la notion de bordure existe : la bordure positive est l'ensemble des motifs fréquents maximaux par rapport à l'inclusion, alors que la bordure négative est l'ensemble des motifs non fréquents minimaux par rapport à l'inclusion.

Un grand nombre de problèmes liés aux fréquents rentrent dans ce cadre tels que les problèmes de découverte de représentations condensées des motifs fréquents. En effet, ils ne correspondent souvent qu'à l'ajout d'une contrainte anti-monotone au prédicat "être fréquent", le tout restant donc anti-monotone. Toutefois, pour certains problèmes, le prédicat n'est plus anti-monotone, et donc la notion de bordure n'est pas applicable. Les motifs fermés fréquents en sont l'exemple typique.

8.1.2 Les dépendances fonctionnelles

Ce cadre peut aussi être appliqué dans un contexte différent des motifs fréquents, tel que la découverte de dépendances fonctionnelles (DF) d'une base de données relationnelle. La découverte de la couverture canonique des DF (ensemble des DF avec un seul attribut en partie droite et partie gauche minimale) est un exemple de problème appartenant à ce cadre théorique.

Découverte des DF de la couverture canonique

Soit une relation r définie sur un schéma de relation R. Les attributs associés à R sont notés $schema(R)$. Soit une dépendance fonctionnelle $X \longrightarrow B$, avec $X \subseteq schema(R)$ et $B \in schema(R)$ fixé. Une telle dépendance est vraie dans r, si pour tout tuple $u, v \in r$, on a : si u et v ont la même valeur pour les attributs de X, alors ils ont la même valeur pour l'attribut B, i.e. $u[X] = v[X] \Rightarrow u[B] = v[B]$.

Ce problème peut être reformulé dans le cadre théorique de la manière suivante [MT97] : Le langage $\mathcal{L}$ est composé de l'ensemble des parties de $schema(R)\backslash\{B\}$, i.e. $\mathcal{P}(schema(R)\backslash\{B\})$. La base de données considérée est la relation r. Le prédicat de sélection $q(r, X)$ est vrai si et seulement si $X \longrightarrow B$ est vrai dans r, avec $X \subseteq schema(R)\backslash\{B\}$. La relation d'ordre est l'inclusion $\subseteq$. La théorie $Th($ r, $\mathcal{P}(schema(R)\backslash\{B\})$, q $)$ est l'ensemble des parties gauches des dépendances fonctionnelles $X \longrightarrow B$, $X \subseteq schema(R)\backslash\{B\}$, satisfaites dans r.

Propriété 14:
Soient r une relation sur R et B un attribut de R. Considérons $q(r, X)$ le prédicat qui teste si la DF $X \longrightarrow B$ est vraie dans r, avec $X \subseteq schema(R)\backslash\{B\}$.
$q(r, X)$ est monotone par rapport à l'inclusion.

La bordure positive est donc composée des plus petites parties gauches vérifiant la dépendance fonctionnelle. La bordure négative est composée des plus grandes parties gauches ne vérifiant pas la dépendance fonctionnelle.

Découverte des clés minimales d'une relation

L'extraction des clés minimales d'une relation est un autre exemple d'application de ce cadre liée aux DF . Rappelons tout d'abord la définition des clés et super clés.

Définition 14:
Soit r une relation sur R, et $X \subseteq schema(R)$. X est une super clé *de r si on a pour tout $u, v \in r, u[X] \neq v[X]$. X est une* clé minimale *de r (ou simplement* clé*) si X est une super clé de r et $\forall Y \subset X, \exists u, v \in r,\ u[Y] = v[Y]$.*

Dans la suite, on note respectivement $SCle(r)$ et $Cle(r)$ l'ensemble des super clés et clés de R. Le problème qui nous intéresse est, à partir d'une relation r sur R, d'extraire toutes les clés minimales de r. Notons que le problème de décision, qui consiste à savoir s'il existe, dans une relation donnée, une clé de taille inférieure à k donné, est NP-Complet [BDFS84].

Dans ce contexte, les mots du langage sont les sous-ensembles d'attributs de $schema(R)$, i.e. $\mathcal{P}(schema(R))$. L'ordre partiel $\preceq$ est trivialement l'inclusion ensembliste $\subseteq$.

Le prédicat "être une clé minimale" n'étant pas monotone ou anti-monotone, étudions le prédicat "être une super clé". Très rapidement, on s'aperçoit que cette notion de super clé est monotone.

Propriété 15:
Soient r une relation et X un ensemble d'attributs de schema(R). Notons $P(X, r)$ le prédicat qui teste si X est une super clé dans r.
$P(X, r)$ est monotone par rapport à $\subseteq$.

Preuve
Nous devons montrer que pour tout X,Y tel que $X \subset Y$, on a P(X,r) vrai $\Rightarrow$ P(Y,r) vrai.
Supposons que P(X,r) est vrai $\Rightarrow \forall u, v \in r,\ u[X] \neq v[X]$.
$\Rightarrow \forall u, v \in r,\ u[X \cup Y \backslash X] \neq v[X \cup Y \backslash X]$.
Donc P(Y,r) est vrai, cqfd.

Les clés minimales d'une relation sont les super clés minimales par inclusion. De plus, $SCle(r)$ est clos par le haut. $Cle(r)$ peut alors être caractérisé à partir de la bordure positive des super clés, i.e. les plus petites super clés par rapport à l'inclusion (le prédicat étant monotone).

Propriété 16:

$$Cle(r) = \mathcal{B}d^{+}(SCle(r))$$

8.1.3 Les dépendances d'inclusion

Les dépendances d'inclusion (DI) sont des contraintes sémantiques fondamentales pour les bases de données relationnelles [MR94, AHV95, LL99]. Malgré la notion d'ordre inhérente aux DI, ce problème est représentable par des ensembles [MFP05].

Soient r et s deux relations, respectivement de schéma R et S. Une dépendance d'inclusion est une expression de la forme $R[X] \subseteq S[Y]$, où X et Y sont des séquences d'attributs de $schema(R)$ et $schema(S)$ respectivement tels que $|X| = |Y|$. La DI $R[X] \subseteq S[Y]$ est satisfaite dans (r, s) si toutes les valeurs des X dans r sont aussi des valeurs de Y dans s. Cette notion généralise les contraintes de clés étrangères.

Ce problème peut être reformulé dans le cadre de découverte de connaissances introduit précédemment [MT97]. Le langage $\mathcal{L}$ est constitué de toutes les dépendances d'inclusion non triviales. Un ordre partiel sur les DI peut être défini de la manière suivante : si i et j sont deux DI, $j \preceq i$ si j peut être obtenu en effectuant la même projection sur les deux parties de i. Par exemple, $R[AB] \subseteq S[EF] \preceq R[ABC] \subseteq S[EFG]$. Le prédicat de sélection $P(r, s, R[X] \subseteq S[Y])$ est vrai si la dépendance $R[X] \subseteq S[Y]$ est satisfaite

dans (r, s). Le problème de fouille de données sous-jacent peut être formulé de la manière suivante : "Soit une base de données, trouver toutes les dépendances d'inclusion satisfaites dans cette base de données" [KMRS92, KR03, MP03].

Propriété 17:
Soient r et s deux relations. Considérons X et Y des sous-ensembles d'attributs de R et S respectivement ($|X|=|Y|$). Notons $P(\ r, s, R[X] \subseteq S[Y]\ )$ le prédicat qui teste si $R[X] \subseteq S[Y]$ est satisfaite dans (r, s).
$P(r, s, R[X] \subseteq S[Y])$ est anti-monotone par rapport à $\preceq$.

Le prédicat étant anti-monotone relativement à l'ordre utilisé, la notion de bordure peut être appliquée pour ce problème. La bordure positive $\mathcal{B}d^+(I)$ est l'ensemble des dépendances d'inclusion satisfaites les plus spécialisées. La bordure négative $\mathcal{B}d^-(I)$ est l'ensemble des dépendances d'inclusion non satisfaites les plus générales par rapport à l'ordre utilisé.

L'obtention d'une représentation ensembliste de ce problème se fait d'après l'intuition suivante : Soit I_1 l'ensemble des dépendances d'inclusion satisfaites de taille 1. La fonction *ens* permet d'associer à chaque dépendance d'inclusion de $\mathcal{L}$, un ensemble de dépendances d'inclusion unaires de la manière suivante : $ens(i) = \{j \in I_1 | j \preceq i\}$.

Par exemple, considérons la dépendance d'inclusion $R[ABC] \subseteq S[FGH]$, avec $\{A, B, C\} \in schema(R)$ et $\{F, G, H\} \in schema(S)$. On a alors $ens(R[ABC] \subseteq S[FGH]) = \{R[A] \subseteq S[F], R[B] \subseteq S[G], R[C] \subseteq S[H]\}$.

En considérant uniquement les DI dont la partie gauche respecte un ordre fixé, on obtient que la fonction *ens* est bijective et admet une fonction inverse ens^{-1}. Dans le cas contraire, la fonction *ens* ne serait pas injective. Par exemple, il serait possible d'avoir $ens(R[AA] \subseteq S[DE]) = ens(R[AA] \subseteq S[ED])$, pour schema(R)={A,B,C} et schema(S)={D,E,F}. Cette transformation est simple mais nécessite de restreindre les DI prises en compte à l'ensemble C défini de la manière suivante [MFP05] :

$$\begin{aligned} C \ = \ & \{i = R[A_1 ... A_n] \subseteq S[B_1 ... B_n] \mid \forall k, l, 1 \leq k < l \leq n, \\ & (A_k < A_l) \vee (A_k = A_l \wedge B_k < B_l) \text{ et i non triviale } \} \end{aligned}$$

Notons que cette restriction n'implique aucune perte de connaissance, grâce au système d'inférence de Mitchell [Mit83].

8.1.4 Réécriture de requêtes

La réécriture de requêtes à partir de vues est une problématique importante, dont les applications vont de l'optimisation de requêtes à l'intégration de sources hétérogènes. Dans [JPR+05], les auteurs ont plus particulièrement étudié ce problème en présence de

contraintes de valeurs, et ont montré que certaines étapes de la réécriture pouvaient se formuler dans le cadre de fouille de données que nous abordons ici.

Le problème de la réécriture de requêtes en termes de vues peut s'énoncer de la façon suivante : Soient Q une requête et $\mathcal{V}$ un ensemble de vues, le problème consiste à reformuler une requête Q en une expression, appelée réécriture, qui utilise uniquement les vues de $\mathcal{V}$ et qui est maximalement contenue dans Q [Hal01, Ull00]. La réécriture de requête peut être vue comme une technique pour calculer les réponses à une requête dans un système de médiation suivant une approche LAV (Local As View). Dans ce type d'approche, les vues, point d'accès aux sources de données, sont décrites comme des requêtes en termes du schéma global du médiateur.

Dans certains cas, l'expression de contraintes de valeurs permet d'obtenir des descriptions de requêtes et de vues très fines. Par exemple, une requête peut demander les parcelles culturales des numéros de commune 23200 ou 23400 qui ont reçu des pesticides durant les années 2001 ou 2002. Les contraintes de valeurs ont ici permis de spécifier les valeurs possibles des attributs numéros de commune et années.

Pour résoudre le problème de la réécriture de requêtes en termes de vues en présence de contraintes de valeurs, les auteurs dans [JPR+05] s'appuient dans un premier temps sur le cadre formel des logiques de description [BCM+03] et adpatent un algorithme de réécriture dans le cadre de la logique étudiée. Ensuite, certaines étapes coûteuses de cet algorithme sont reformulées en problèmes d'extraction de motifs intéressants, permettant ainsi d'exploiter les algorithmes existants dans ce domaine et d'optimiser ces étapes.

Notons que pour cet exemple le niveau de granularité est différent des exemples précédents. Seules certaines étapes de la méthode de résolution peuvent être reformulées en problèmes d'extraction de motifs intéressants, et non l'ensemble du problème comme cela est le cas pour les fréquents, les DF ou les DI. Toutefois, ces étapes constituent un point difficile de la méthode en termes de passage à l'échelle. Leur reformulation dans ce cadre théorique permet donc d'exploiter les méthodes efficaces développées pour l'extraction de motifs.

8.2 Application des algorithmes existants à l'extraction de motifs intéressants

Dans cette section, nous allons nous intéresser à la question de l'application des algorithmes existants d'extraction de motifs fréquents dans le cadre théorique de [MT97]. Tout d'abord, nous allons simplement étudier s'il est possible d'utiliser ces algorithmes dans ce nouveau contexte, et sous quelles conditions. Ensuite, nous verrons l'impact que cela a sur les algorithmes.

Comme nous l'avons vu précédemment, la majorité des algorithmes utilisent des pro-

priétés liées au support pour améliorer leur efficacité. Néanmois, il est souvent possible de les supprimer et d'utiliser uniquement la stratégie "basique" d'exploration de l'espace de recherche de l'algorithme. Les algorithmes $Mafia$ [BCG01] et $GenMax$ [GZ01] sont par exemple dans ce cas. Ils utilisent des optimisations propres aux fréquents, mais peuvent être réduits au parcours de l'espace de recherche qu'ils effectuent, i.e. un parcours en profondeur avec élagage à partir des motifs de la bordure positive découverts (section 4.2.1).

Toutefois, ce n'est pas toujours le cas. Une stratégie telle que celle utilisée par $FP-growth$ [HPY00] est totalement dépendante du problème des motifs fréquents (section 4.2.2). De la même manière, LCM [UAUA03] ne peut être appliqué que si la notion de fermeture existe (section 4.2.2).

De plus, ces adaptations risquent de beaucoup affecter les performances des algorithmes, et peuvent même influencer la stratégie d'exploration de l'espace de recherche.

En effet, leurs performances dépendent fortement d'optimisations spécifiques au prédicat "être fréquent", notamment pour améliorer le calcul du support (sections 5.2 et 7.4). Par conséquent, leur efficacité pour un autre prédicat est plus qu'incertaine. Les algorithmes les plus performants pour l'extraction des motifs fréquents ne le sont donc pas nécessairement pour un problème équivalent.

Les stratégies d'exploration de l'espace de recherche peuvent aussi être affectées. Par exemple, les sauts effectués par une grande partie de ces algorithmes sont généralement approximatifs et dépendent de l'ordre dans lequel les motifs sont considérés. Pour le problème de l'extraction des motifs fréquents, les motifs sont souvent ordonnés par support croissant. Comme l'ont montré différents travaux [AAP01, Bor03, Bod04], cet ordre permet en pratique d'améliorer fortement les performances des algorithmes. Toutefois, cette heuristique est propre au problème des fréquents, et peut difficilement être appliquée dans un autre contexte.

De manière générale, peu d'algorithmes s'appuient sur une stratégie totalement générique pouvant être appliquée directement à n'importe quel problème d'extraction de motifs intéressants. A titre d'exemple, des algorithmes tels qu'*Apriori* [AS94], *Eclat* [ZPOL97] ou *Dualize and Advance* [GKM$^+$03] sont dans ce cas. Comme nous allons le voir maintenant, l'algorithme ABS [FMP04] que nous avons proposé fait aussi partie de cette famille d'algorithmes totalement génériques.

Généricité d'ABS L'algorithme ABS s'appuie sur deux types de stratégie : la première est un parcours par niveau identique à celui utilisé dans *Apriori* (section 4.1.1), et la deuxième est basée sur la notion de dualisation (section 4.3). Or comme cela a été montré dans [MT97], *Apriori* peut être généralisé à tout problème d'extraction de motifs intéressants, sans modification de sa stratégie. De même, les dualisations effectuées ensuite par ABS s'appuient sur les théorèmes définis en section 3 pour l'ensemble des problèmes d'extraction de motifs intéressants représentables par des ensembles.

ABS est aussi fondé sur deux aspects adaptatifs (section 7.1.4). Le premier, et le plus important, permet de déterminer la fin de l'approche par niveau et le début de la

phase de dualisation entre les deux bordures. Pour le problème des motifs fréquents, ce changement de stratégie est déterminé d'après la proportion de motifs fréquents générés lors de la dernière itération d'*Apriori* (section 7.1.4). Cette stratégie adaptative peut donc simplement être généralisée en étudiant la proportion de motifs intéressants générés par l'approche par niveau.

Le second aspect adaptatif exploite l'éloignement d'un motif par rapport à la bordure positive (section 7.1.4). Bien que la définition de la mesure soit fortement dépendante du problème étudié, cette notion est générique et peut être redéfinie pour tout problème d'extraction de motifs intéressants. Par exemple, dans le cadre de l'étude des DI les plus spécialisées, une mesure a été définie dans [LPT02] et permet ainsi d'estimer la distance entre une DI non satisfaite et la bordure positive.

Pour finir, notons que l'objectif de cet algorithme est de limiter le nombre de motifs candidats testés par rapport au prédicat, et non d'optimiser le traitement du prédicat. Son efficacité est donc totalement indépendante du prédicat considéré, contrairement aux meilleurs algorithmes d'extraction de motifs fréquents.

L'efficacité de ce type de stratégie a été démontrée dans [MP03] pour le problème de l'extraction des DI. Pour ce problème, le coût du test du prédicat est important, ce qui explique l'intérêt de notre approche dont l'objectif est de limiter le nombre de motifs testés. Cet intérêt est aussi en partie lié à une propriété théorique des DI qui se traduit par des distributions des bordures particulières, correspondant au type II (grande distance entre les bordures) et au type III (bordures proches mais composées de petits éléments) de la classification proposée dans le chapitre 6.

Notons que les résultats du chapitre 6 peuvent ainsi être généralisés aux problèmes d'extraction de motifs intéressants. Pour ce type de problèmes, l'étude de la bordure positive et de la bordure négative d'un prédicat anti-monotone (ou monotone) donné peut nous permettre d'avoir des informations très générales sur ce prédicat. Cette étude peut notamment nous permettre de mieux appréhender le comportement des différentes stratégies existantes, et donner des indications sur l'existence de propriétés associées au prédicat étudié.

8.3 Les solutions logicielles existantes pour l'extraction de motifs

Une grande partie des implémentations des algorithmes d'extraction de motifs fréquents se présente sous la forme de programmes indépendants, développés principalement pour évaluer l'efficacité de l'algorithme. C'est notamment le cas de toutes les implémentations rendues disponibles par les ateliers FIMI.

D'autres travaux tels que *Illimine* [HG05] ou *Weka* [WF05], proposent des outils de fouille de données et d'apprentissage libres de droits intégrant des algorithmes d'extraction

de motifs fréquents.

Le projet *Illimine* est une librairie C/C++ développée par l'équipe de recherche du professeur J. Han, et regroupant les implémentations des algorithmes proposés par celle-ci. Les domaines actuellement abordés par ce projet sont le cube de données, la découverte des règles d'associations, l'extraction des motifs séquentiels, la fouille de graphes et la classification. De plus, des outils ont été développés au sein de ce projet afin de faciliter l'utilisation des exécutables.

De son côté, *Weka* est un logiciel gratuit d'apprentissage et de fouille de données parmi les plus utilisés. Ce logiciel, développé en Java et constitué d'une interface graphique, regroupe une collection d'algorithmes d'apprentissage et de fouille de données. Il contient des outils pour le pré-traitement des données, la classification, la régression, la découverte de règles d'associations et la visualisation. A partir de l'interface graphique, ce logiciel permet aussi de développer de nouveaux processus d'extraction de connaissances. Plus basiquement, *Weka* peut aussi être utilisé telle une librairie de composants Java.

Toutefois, ces travaux ne font que regrouper des implémentations d'algorithmes dédiées et optimisées pour la résolution de problèmes précis. Ces programmes ne peuvent donc pas être utilisés pour résoudre d'autres problèmes même s'ils sont équivalents. Les codes sources ne sont pas toujours disponibles, et lorsqu'ils le sont leur modification s'avère difficile voire impossible de par la complexité de l'implémentation.

Notons que des solutions commerciales existent aussi. Par exemple, *SAS Enterprise Miner* permet d'appliquer différents traitements de fouille de données, dont l'extraction de motifs fréquents. Mais l'utilisation des algorithmes intégrés au logiciel pour résoudre tout autre problème que celui prévu initialement est impossible, d'autant plus que le code source n'est pas disponible.

A notre connaissance, un seul travail tend à proposer une solution "générique" visant à simplifier la réutilisation des algorithmes d'extraction de motifs fréquents : la librairie DMTL (Data Mining Template Library) [HCS+05]. Elle permet à l'utilisateur d'adapter les algorithmes implémentés par l'équipe de M.J. Zaki afin de résoudre des problèmes de découverte de motifs (au sens large) fréquents.

La librairie DMTL DMTL est une collection d'algorithmes et de structures de données. Les structures de données génériques définies permettent de traiter des motifs variés, et de concevoir des algorithmes génériques d'extraction de motifs fréquents. La librairie permet d'accéder à différents types de base de données en mémoire ou sur le disque, et de les gérer de manière horizontale ou verticale (section 5.2).

Dans leur librairie, les auteurs représentent les différents types de motifs sous forme de graphes. La figure 8.1 illustre des motifs tels que les ensembles, les séquences, les arbres et les graphes. Ainsi, les ensembles d'articles sont par exemple des graphes ayant pour noeuds les articles et aucune arête.

Concrètement, une même structure de données est utilisée pour représenter tous les

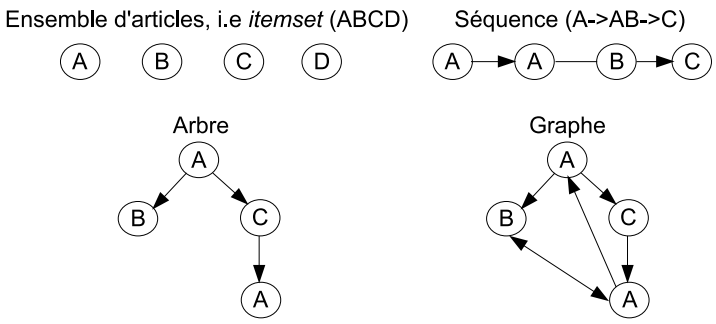

FIG. 8.1 – Représentation des motifs dans la librairie DMTL [ZPD+05]

motifs sous forme de graphe, et chaque type de motifs est différencié par un ensemble de propriétés et de méthodes :

- propriétés sur la structure des motifs. Elles permettent de définir quel type de graphe va représenter les motifs, i.e. les propriétés sur les arêtes et les noeuds. Par exemple, les arbres sont caractérisés par les propriétés "connexe" et "acyclique", car par définition ce sont des graphes connexes et acycliques.
- méthodes utilisées lors de l'extraction, mais propres au motif étudié. Par exemple, il est nécessaire de définir les méthodes de génération des candidats (section 4.1) et de comptage du support. En effet, celles-ci sont totalement différentes d'un type de motif à un autre, par exemple entre les ensembles et les graphes.

Dans ce contexte, l'ajout d'un nouveau type de motif à la librairie correspond simplement à l'ajout de nouvelles propriétés et méthodes.

D'un point de vue conceptuel, les auteurs utilisent le cadre théorique de l'*analyse formelle de concepts* [GW99] pour modéliser la hiérarchie des différents types de motifs. Plus concrètement, les motifs et leurs propriétés sont représentés par des concepts formels (A, B), où A est une collection de différents types de motifs et B un ensemble de propriétés communes à tous les motifs de A. La hiérarchie entre les différentes classes de motifs et leur propriétés est modélisée par le treillis des concepts. La figure 8.2 est la représentation simplifiée de ce treillis, où seul apparaissent de bas en haut les nouveaux motifs et de haut en bas les nouvelles propriétés. Par exemple, le noeud contenant $\{\{\text{Graphe acyclique orienté}\}, \{\emptyset\}\}$ représente le concept {{Graphe acyclique orienté, Séquence, Arbre ordonné, Arbre non ordonné}, {Acyclique,Arêtes orientées}}. Les auteurs se sont appuyés sur cette hiérarchie pour développer des composants génériques pouvant traiter différents types de motifs. En effet, tout composant fonctionnant pour un motif appartenant à un concept de la hiérarchie doit aussi fonctionner automatiquement pour tous motifs des sous-concepts. Par exemple, les composants développés pour les graphes orientés acycliques doivent fonctionner pour les arbres non ordonnés, les arbres ordonnés et les séquences. Par contre, ils ne seront pas nécessairement très efficaces, d'où l'intérêt de redéfinir tout de même certains composants pour ces "sous-motifs".

La librairie contient actuellement deux types de stratégies d'exploration de l'espace

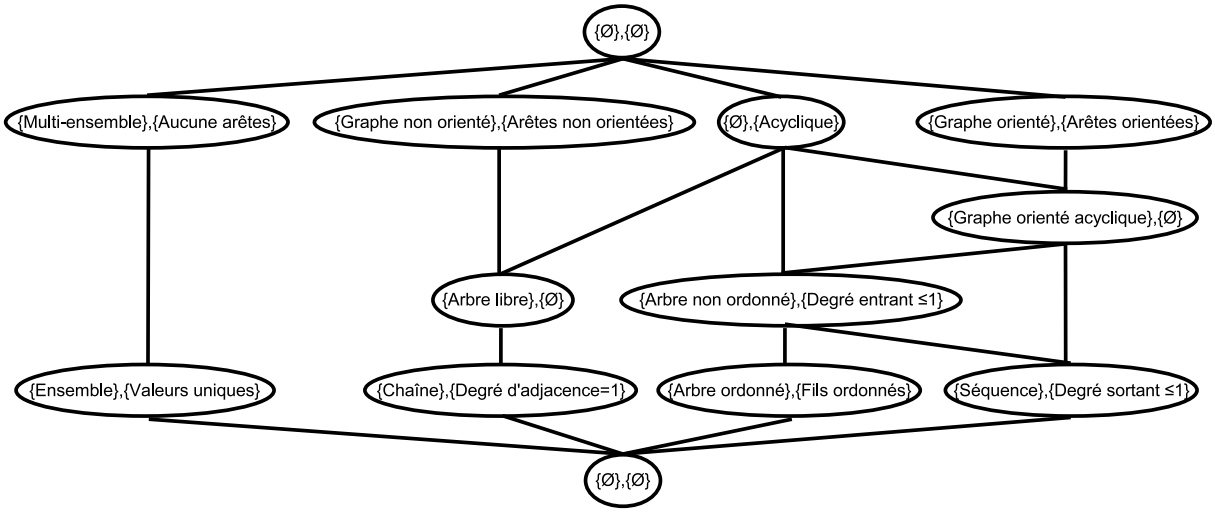

FIG. 8.2 – Hiérarchie des motifs-propriétés dans $DMTL$

de recherche : une stratégie effectuant un parcours en largeur (ou par niveau) de type *Apriori* et une stratégie effectuant un parcours en profondeur. Ces stratégies permettent de trouver l'ensemble de motifs fréquents de la base de données étudiée.

A nos yeux, les inconvénients de cette librairie dans son état actuel de développement sont :

- seul le prédicat de la fréquence peut être pris en compte. $DMTL$ ne semble pas tirer profit de la caractérisation des problèmes représentables par des ensembles de [MT97].
- seule l'énumération de tous les motifs fréquents est possible. Les bordures ne peuvent pas être recherchées.
- seuls sont disponibles des algorithmes "classiques", testant un grand nombre de motifs. Aucune des stratégies n'exploite par exemple les motifs de la bordure positive découverts pour élaguer l'espace de recherche.

Chapitre 9

Vers le prototypage rapide d'algorithmes d'extraction de motifs intéressants

Sommaire

Dans ce chapitre, notre objectif est de proposer un outil facilitant le développement de solutions logicielles efficaces utilisant les algorithmes d'extraction de motifs fréquents pour résoudre des problèmes équivalents d'extraction de motifs (au sens large) intéressants, représentables par des ensembles [MT97].

Nous proposons donc une librairie [FMP06] facilitant la résolution de tels problèmes, basée sur l'utilisation d'algorithmes et de structures de données génériques, et passant à l'échelle. Les caractéristiques et optimisations de l'implémentation sont transparentes pour le programmeur, et seules les propriétés spécifiques à son problème sont laissées à sa charge.

De plus, nous proposons une ébauche de méthodologie pour guider l'utilisateur dans ses choix. Elle aide l'utilisateur à reformuler (si possible) son problème sous la forme d'un problème d'extraction de motifs intéressants, et l'assiste aussi dans le développement des

différents composants.

9.1 Cadre théorique

Le cadre théorique utilisé est celui des problèmes, *représentables par des ensembles*, d'extraction de motifs intéressants dans les bases de données [MT97] présenté dans le chapitre 3.

Ce cadre nous a notamment déjà permis d'appliquer des algorithmes issus de l'extraction de motifs fréquents à d'autres problèmes équivalents, tels que pour la découverte des DF [LPL00], la découverte des DI les plus spécialisées [MLP02, MP03] et la résolution de certaines phases d'un problème de réécriture de requêtes [JPR+05]. Dans ce dernier, l'implémentation développée a ainsi permis le passage à l'échelle en fonction du nombre de vues traitées [JF06].

L'obtention d'une représentation ensembliste peut sembler être une contrainte forte, mais elle a beaucoup d'avantages et permet d'envisager une généricité importante. Par exemple, lors d'un parcours "par niveau" de l'espace de recherche, la génération des candidats du niveau k à partir des motifs vrais de niveau $k-1$ peut être faite efficacement en utilisant un cadre ensembliste, dont l'archétype est *AprioriGen* [AS94]. Avoir un problème représentable par des ensembles permet donc de facto d'utiliser cette génération des candidats , d'où une grande généricité. Par conséquent, l'utilisateur doit seulement définir le prédicat, les données en entrée/sortie et éventuellement une fonction de transformation des motifs vers le treillis des parties d'un ensemble. Il n'a donc pas à implémenter les parties les plus complexes, tels l'algorithme d'énumération utilisé ou la gestion des candidats en mémoire.

De plus, beaucoup d'algorithmes s'appuient sur des méthodes nécessitant une représentation ensembliste du problème comme par exemple l'algorithme *Dualize and Advance* (section 4.3) ou *ABS* (chapitre 7).

9.2 Aspects méthodologiques

Cette section apporte des éléments méthodologiques pour permettre à un analyste d'utiliser concrètement ce que nous proposons.

9.2.1 Reformulation d'un problème dans le cadre

Dans cette section, nous allons montrer un cheminement possible à effectuer pour vérifier qu'un problème appartient à la famille de problèmes traités par notre librairie. Ce

cheminement est composé des quatre étapes suivantes :
- définir l'espace de recherche,
- définir le prédicat,
- identifier la sortie,
- exhiber une représentation ensembliste.

L'exemple de l'extraction des DI satisfaites entre deux relations r et s, vu section 8.1.3, est repris comme exemple dans cette section.

Définir l'espace de recherche

Dans un premier temps, il faut déterminer le langage qui permet d'exprimer les motifs puis une relation d'ordre partiel, pour définir l'espace de recherche.

Exemple 5:
Sur l'exemple des DI, les motifs du langage sont les DI $R[X] \subseteq S[Y]$ *sur les relations* r *et* s *de schéma* R *et* S*, où* $X \subseteq schema(R)$*,* $Y \subseteq schema(S)$*, et* $|X| = |Y|$*.*

L'ordre partiel $\preceq$ *est défini de la manière suivante : si* i *et* j *sont deux DI,* $j \preceq i$ *si* j *peut être obtenu en effectuant la même projection sur les deux parties de* i*.*

Définir le prédicat

Identifier le prédicat est souvent aisé, il suffit d'écrire formellement ce que "intéressant" signifie dans le contexte. Pour un motif et une base de données, il faut écrire la condition qui rend ce motif intéressant dans cette base de données.

Exemple 6:
Pour notre problème, le prédicat permettra de répondre oui ou non à la question suivante : $R[X] \subseteq S[Y]$ *est-elle une DI satisfaite dans* r *et* s *?*

L'utilisateur doit alors démontrer la propriété d'anti-monotonie (ou de monotonie) liant la satisfaction du prédicat et l'ordre partiel entre les motifs.

Exemple 7:
Il nous faut démontrer que le prédicat "être une DI satisfaite" est monotone ou anti-monotone. Comme nous l'avons vu en section 8.1.3, ce prédicat est anti-monotone.

De façon générale, notons que certains prédicats ne vérifient pas cette propriété de monotonie ou d'anti-monotonie, typiquement pour certaines mesures statistiques [MS00].

Identifier la sortie

Il s'agit ici de caractériser quels motifs doivent être donnés en sortie, en fonction de l'ensemble des motifs intéressants. Nous verrons plus loin que cette caractérisation est importante pour guider l'analyste dans son choix de la stratégie d'exploration.

Exemple 8:
Selon les besoins de l'analyste, la solution de l'extraction des DI satisfaites dans r et s pourra simplement être la théorie. Si l'utilisateur souhaite une couverture des DI satisfaites dans r et s, la bordure positive de la théorie sera alors la sortie de l'algorithme.

Exhiber une représentation ensembliste

De nombreux problèmes s'expriment déjà naturellement de façon ensembliste, et pour eux cette étape est triviale. A notre connaissance, le seul problème pour lequel une transformation non triviale a été donnée est celui de la découverte des dépendances d'inclusion dans les bases de données. Nous pensons que, lorsqu'elle n'est pas triviale, cette étape peut s'avérer très délicate.

Exemple 9:
Comme nous l'avons vu en section 8.1.3, la fonction permettant de transformer une DI en ensemble est la suivante : $ens(i) = \{j \in I_1 | j \preceq i\}$, avec I_1 l'ensemble des dépendances d'inclusion satisfaites de taille 1.

9.2.2 Les stratégies d'exploration disponibles

Les algorithmes

Une fois le problème placé dans ce cadre, le développeur doit choisir une stratégie d'exploration de l'espace de recherche. Actuellement, deux options sont disponibles :

- un algorithme par niveau de type *Apriori* [AS94] (section 4.1.1) qui permet de découvrir la théorie et les deux bordures.
- l'algorithme *ABS* [FMP04] (chapitre 7) qui permet uniquement de trouver les deux bordures.

Ces algorithmes peuvent traiter des prédicats anti-monotones ou monotones. Ce dernier cas est possible à condition d'étudier le prédicat inverse et de rechercher uniquement les bordures. En effet, si un prédicat P est monotone alors le prédicat inverse $\neg P$ est anti-monotone, et la bordure positive (resp. bordure négative) de P est la bordure négative (resp. bordure positive) de $\neg P$.

Afin de montrer l'exécution de ces algorithmes lorsque le prédicat étudié est monotone, les exemples suivants présentent les différentes stratégies disponibles dans le cadre de l'extraction des clés minimales d'une relation (section 8.1.2).

Exemple 10:
Pour découvrir la bordure positive des super clés (i.e. les clés minimales), il faut donc étudier le prédicat "ne pas être une super clé", défini par $\neg P(X, d)$ *est vrai si et seulement si* $|\pi_X(r)| \neq |r|$ *, et rechercher la bordure négative liée à ce prédicat.*

Pour faire cela, nous avons le choix entre les deux types de parcours présentés précédemment.

Considérons la relation r suivante :

A	B	C	D	E
0	1	b	2	e
1	2	b	1	f
1	1	a	3	a
1	1	a	4	e
1	1	a	5	a
0	1	a	6	a
1	2	b	6	a

TAB. 9.1 – Relation r

La figure 9.1 montre le parcours par niveau effectué par l'algorithme Apriori. Les éléments foncés sont les éléments générés et testés par l'algorithme. Les chiffres en dessous sont les cardinalités des projections calculées par l'algorithme lors du test du prédicat. Les éléments générés et testés sont tous "non super clés" à l'exception de $\{AD, CD, BD\}$*. La partie en clair correspond à l'espace de recherche qui n'a pas été exploré par l'algorithme. L'algorithme trouve la bordure positive, i.e.* $\{ABCE, DE\}$*, en quatre itérations. Les clés minimales sont donc* $\{AD, CD, BD\}$*.*

La figure 9.2 montre le parcours effectué par ABS. Supposons que l'algorithme explore les deux premiers niveaux lors de sa phase d'initialisation. En s'appuyant sur les mauvais éléments découverts, ici $\{AD, CD, BD\}$*, l'algorithme effectue une dualisation et trouve directement la bordure positive (*$\{ABCE, DE\}$*).*

Comme nous l'avons vu précédemment, l'intérêt de ces algorithmes est de pouvoir être utilisés directement pour résoudre tout problème d'extraction de motifs intéressants, représentable par des ensembles, contrairement à une grande partie des autres algorithmes. De plus, leur objectif étant de limiter le nombre de motifs testés et non d'optimiser le comptage du support, ils sont donc particulièrement adaptés dans notre contexte.

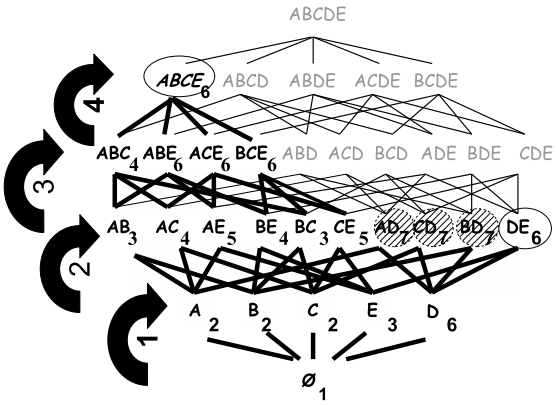

FIG. 9.1 – Exécution d'Apriori

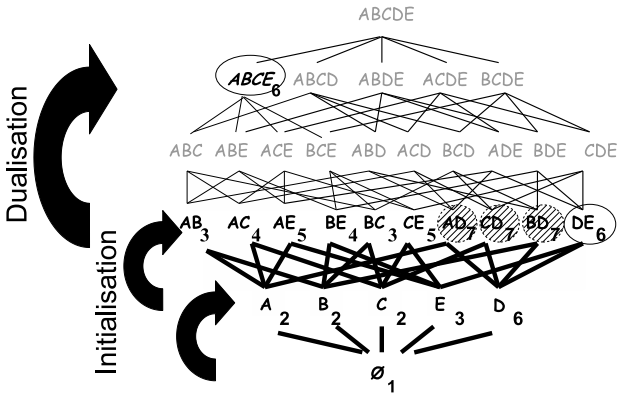

FIG. 9.2 – Exécution d'ABS

Changement du sens de parcours des algorithmes

Pour chaque algorithme, il est aussi possible de changer le sens de parcours de l'espace de recherche :

- Avec *Apriori*, l'espace de recherche est exploré de "haut en bas".
- Avec *ABS*, l'initialisation commence "par le haut" et alterne des dualisations entre les deux bordures comme précédemment.

Une approche "par le haut" permet notamment de trouver la théorie des prédicats monotones. Dans le cas d'un prédicat anti-monotone, il peut s'avérer pertinent en fonction de la nature des données d'utiliser une approche "par le haut" : il faut alors aussi inverser le prédicat, et accepter de ne trouver que les bordures de la théorie.

Notons que pour inverser le sens de parcours, il suffit d'utiliser les algorithmes classiques et de transformer tous les motifs testés par rapport au prédicat en leur complémentaire. Cette simple opération a pour conséquence d'inverser le sens de parcours de l'algorithme sans modifier la complexité de l'algorithme, ni son efficacité.

Exemple 11:

Considérons une nouvelle relation s :

F	G	H	I	J
0	a	d	2	e
0	b	d	2	0
0	a	a	2	1
0	a	d	4	e
0	a	d	2	f

TAB. 9.2 – Relation s

Le prédicat "être une super clé" est monotone. Il suffit donc de transformer chaque motif étudié par l'algorithme en son complément et d'appliquer le prédicat sur ce complément. Par exemple, lors de la première itération d'Apriori "inversé", l'algorithme va donc étudier les motifs $\{F, G, H, J, I\}$*, mais tester les motifs* $\{GHIJ, FHIJ, FGIJ, FGHI, FGHJ\}$*. Les motifs* $FGHJ$ *et* $FGHI$ *n'étant pas intéressants par rapport au prédicat, l'algorithme considère les motifs* I *et* J *comme faux. L'itération suivante va donc utiliser les motifs* $\{F, G, H\}$ *et générer les motifs candidats* $\{FG, FH, GH\}$ *(leurs sous-ensembles étant considérés comme intéressants). La deuxième itération va donc tester par rapport au prédicat les motifs* $\{HIJ, GIJ, FIJ\}$*, et ainsi de suite jusqu'à ce que plus aucun motif candidat ne soit généré.*

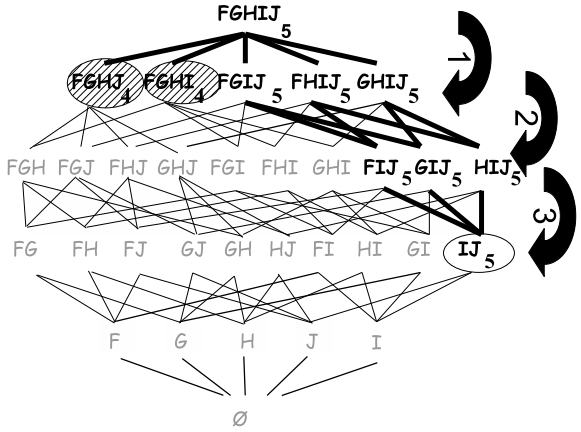

FIG. 9.3 – Exécution d'Apriori en changeant le sens de parcours

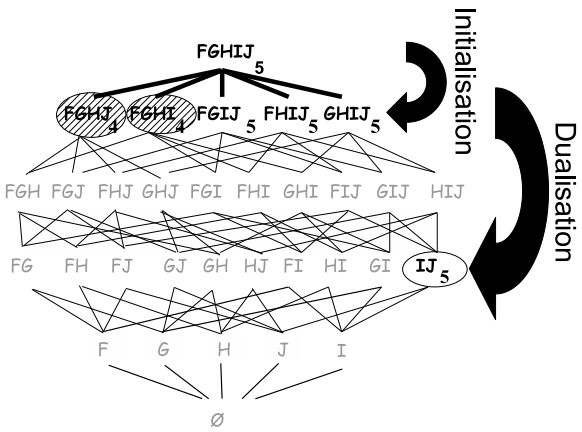

FIG. 9.4 – Exécution d'ABS en changeant le sens de parcours

La figure 9.3 montre le parcours par niveau effectué par l'algorithme Apriori lorque le sens de parcours est inversé. Le parcours se fait de haut en bas à partir des super clés découvertes et en élaguant les motifs non super clés.

La figure 9.4 montre le parcours effectué par ABS en changeant le sens de parcours et en explorant un niveau pour la phase d'initialisation.

Choix de la stratégie d'exploration de l'espace de recherche

La librairie présentée offre à l'utilisateur une grande liberté dans l'utilisation des algorithmes, ce qui lui permet de choisir la stratégie la plus adaptée à son problème. Le choix du mode de parcours de l'espace de recherche dépend en partie de la sortie que l'on souhaite obtenir, mais aussi des caractéristiques du problème étudié et/ou des données.

Par exemple, *ABS* sera particulièrement adapté à l'énumération des plus grandes DI satisfaites dans une base de données (section 8.2). En revanche, comme l'ont montré les ateliers FIMI, *Apriori* est très compétitif dès lors que les données sont creuses et que les solutions sont de "petite" taille.

Exemple 12:
Dans l'exemple 10, l'algorithme ABS teste moins de motifs que l'algorithme Apriori. Pour ce jeu de données, ABS devrait être plus efficace.

Lorsque beaucoup de super clés sont de grande taille et peu de petite taille, comme c'est le cas dans l'exemple 11, changer le sens de parcours des algorithmes est une stratégie intéressante. En effet, les algorithmes testent moins d'éléments en effectuant un parcours de haut en bas qu'un parcours de bas en haut.

D'autres parcours génériques, par exemple en profondeur, peuvent bien sûr être considérés, chacun d'entre eux présentant des avantages suivant les configurations. Notre librairie est largement extensible, nous verrons plus loin le diagramme de classe d'un algorithme implanté dans la librairie.

9.3 Une librairie C++

L'idée poursuivie dans ce travail est de fournir une boîte à outils qui permette de développer rapidement un code efficace et robuste en utilisant tel ou tel algorithme préalablement choisi.

9.3.1 Organisation

La librairie est constituée d'algorithmes et de composants utilisés par les algorithmes pour pouvoir fonctionner. Elle met aussi à disposition de l'utilisateur des structures de données et d'autres composants pour faciliter son utilisation.

Chaque algorithme est représenté par une classe qui contient la stratégie de l'agorithme indépendamment du prédicat étudié. Pour l'utilisateur, cette classe est une "boîte noire". Il interagit uniquement avec les composants liés aux prédicats. Selon le problème étudié, l'utilisateur devra développer certaines de ces classes, ou pourra plus simplement utiliser les classes déjà développées pour un autre problème.

La figure 9.5 représente le diagramme de classe d'un algorithme au sein de la librairie. Ce diagramme de classe est directement issu du cadre théorique utilisé et des applications étudiées. Il peut notamment être utilisé pour développer et intégrer de nouvelles stratégies dans la librairie.
Selon le problème étudié, jusqu'à cinq types de classes sont à redéfinir par l'utilisateur : les données en entrée de l'algorithme, la fonction d'initialisation du langage, le prédicat, les données en sortie de l'algorithme et la fonction de transformation des motifs du langage. Certaines de ces classes doivent nécessairement être définies, telles que le prédicat et la fonction d'initialisation du langage, les autres dépendent du problème étudié.

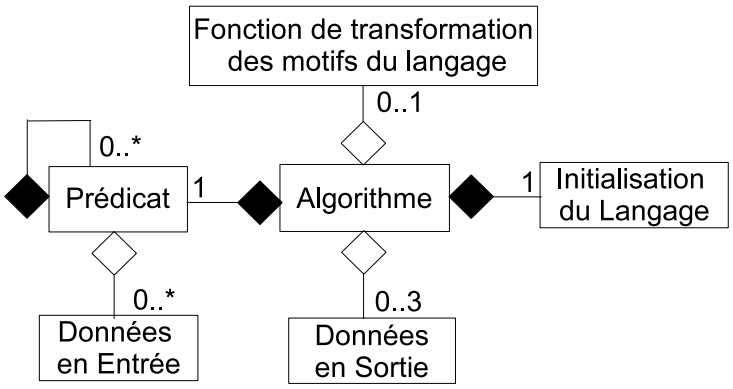

FIG. 9.5 – Diagramme de classe d'un algorithme dans la librairie

9.3.2 Son utilisation

La suite décrit le fonctionnement de notre librairie en présentant les classes à redéfinir (i.e. à implémenter ou à réutiliser) pour chaque problème et comment utiliser les algorithmes implémentés afin de résoudre un nouveau problème.

Les composants à redéfinir

Tout d'abord, l'utilisateur doit redéfinir les données en entrée de l'algorithme, i.e. celles utilisées par le prédicat pour déterminer les motifs "intéressants". Comme le montre la figure 9.5, ce composant est uniquement utilisé dans le test du prédicat, il est totalement indépendant de l'algorithme. L'utilisateur peut donc développer de nouveaux composants, utiliser ceux de la librairie, ou ceux de la STL, sans aucune restriction. Au delà de l'implémentation, cette organisation permet de gérer tout type de données tels que des données en mémoire, un fichier texte, ou une base de données.

Ensuite, il faut définir la classe "initialisant le langage" dont l'objectif est d'initialiser "l'alphabet" du langage, i.e. les motifs de rang 1. Par exemple, si le problème étudié est la découverte des clés minimales, cette phase correspond simplement à la lecture du schéma de la relation. Pour le problème de l'extraction des DI satisfaites entre deux relations, cette phase correspond à la lecture du schéma des deux relations, et à la génération des DI unaires du langage.

L'utilisateur doit ensuite implémenter la classe prédicat (figure 9.5), utilisée dans l'algorithme pour tester si un motif respecte le prédicat. Cette classe peut manipuler plusieurs données en entrée. Par exemple, dans le cadre de la découverte des DI entre deux relations, le prédicat référencera les deux relations concernées.

Dans certains cas, les motifs sont associés à une mesure d'intérêt tel que le support pour les motifs fréquents. Le prédicat peut alors être utilisé pour calculer cette mesure et l'associer aux motifs.

Finalement, notons qu'il est possible de définir des prédicats issus de la conjonction de plusieurs prédicats, tels que les prédicats des représentations condensées des motifs fréquents. Dans ce cas, l'organisation de notre librairie permet de réutiliser les prédicats déjà développés, et de les associer très simplement dans un nouveau prédicat. Par exemple pour les motifs générateurs fréquents, si le prédicat "être fréquent" est déjà défini, il suffit d'implémenter celui définissant les motifs générateurs, puis de les intégrer tous deux dans un nouveau prédicat "être générateur fréquent".

L'utilisateur doit aussi développer ou utiliser des classes pour stocker et/ou enregistrer les solutions découvertes par l'algorithme (la théorie, la bordure positive et/ou la bordure négative). Ces classes peuvent être utilisées pour stocker les solutions en mémoire, pour les enregistrer dans un fichier dans un format donné ou sur un autre support. La définition de ce composant permet également d'effectuer un éventuel post-traitement sur la sortie tels que le filtrage de certains éléments, l'application d'un critère de maximalité ou d'un ordre de sortie particulier.

Finalement, pour tout problème ne s'exprimant pas naturellement de façon ensembliste, il est nécessaire de définir une fonction permettant de transformer chaque motif du langage en ensemble. Par exemple, dans le cas de l'extraction des DI satisfaites entre deux relations, cette fonction va correspondre à la fonction *ens* définie en section 8.1.3. Elle prendra en paramètre une DI et retournera l'ensemble des DI unaires correspondant.

Quelques problèmes déjà implantés

Dans sa version actuelle, notre librairie contient des composants pour la découverte des motifs fréquents, des motifs essentiels fréquents, la découverte des clés minimales d'une relation, et la découverte des DI satisfaites entre deux relations. Ils constituent une boîte à outils mettant à la disposition de l'analyste :

- Des composants permettant d'accéder à des données stockées dans des fichiers : base de données de transactions au format de FIMI [BZ03, BGZ04], ou données tabulaires au format CSV d'Excel.
- Des structures de données de type "*trie*", particulièrement adaptées pour représenter des ensembles.
- Les prédicats "être une super clé", "être une DI satifaite", "être fréquent", et "être essentiel fréquent".

Appel d'un algorithme

Pour utiliser un algorithme, il suffit d'instancier la classe correspondante en lui passant en paramètre les composants nécessaires à la résolution du problème (parmi ceux présentés précédemment). En outre, si besoin est, la librairie intègre un composant permettant d'inverser simplement un prédicat.

9.3.3 Performances et optimisations

Les performances des composants utilisateurs dépendent directement de la façon dont ils ont été implémentés. Toutefois, un certain nombre de composants sont disponibles afin de faciliter l'obtention d'une implémentation efficace. Par exemple, la librairie comprend deux structures de "*trie*" : la première favorise la compression des données, alors que la deuxième favorise les temps d'accès aux données.

Par ailleurs, le développement d'algorithmes modulaires et génériques implique nécessairement un surcoût à l'exécution par rapport à des solutions dédiées, essentiellement dû à des passages de paramètres supplémentaires, et à l'utilisation d'itérateurs.

Un certain nombre de techniques d'optimisations dédiées peuvent en outre influer fortement sur les performances. La découverte des motifs fréquents est un exemple typique, où tous les algorithmes utilisent des techniques astucieuses de comptage. En général, ces optimisations sont liées au test du prédicat, et peuvent être perçues comme des pré-traitements ou des post-traitements du prédicat. Pour cette raison, deux méthodes ont été ajoutées au prédicat : "preProcessing" et "postProcessing". Ces deux méthodes sont exécutées dans les algorithmes avant et après la phase de test des candidats. Elles prennent en paramètre l'ensemble des candidats, ainsi que la fonction de transformation des motifs du langage. Par exemple, considérons la découverte des motifs fréquents :

- la méthode "preProcessing" peut par exemple servir à mettre à jour le support des candidats avant d'effectuer le test du prédicat. Ainsi il est possible de faire un parcours de la base de données à chaque itération de l'algorithme au lieu d'un parcours pour chaque motif testé (comme cela est fait dans l'adaptation à ce cadre de l'algorithme *Apriori* [MT97]).
- la méthode "postProcessing" peut être utilisée pour reconstruire la base de données en supprimant des articles ou des transactions qui ne sont plus utilisés.

```
Class isFrequent{

    ...

    template< class Data, class Candidates>
    count( Data , Candidates ){ ... }


    template< class Candidates>
    count( TrieData , Candidates ){ ... }

};
```

FIG. 9.6 – Exemple d'implémentation du prédicat "être fréquent" contenant une méthode générique pour le comptage du support et une spécifique lorsque les données sont stockées dans un trie

Il est également possible d'améliorer considérablement les performances des implémentations en tirant partie des spécificités de certaines structures de données, comme les "*tries*" dans le cas des motifs fréquents. Comme le montre la figure 9.6, il est possible de développer au sein d'une même classe plusieurs versions d'une même méthode : une version générique et des versions spécialisées. La méthode générique est automatiquement utilisée si aucune méthode spécialisée ne correspond à la structure de données traitée.

9.4 Expérimentations

L'évaluation objective du temps de développement est très difficile. Le seul recul que nous avons par rapport à cet aspect provient de nos propres travaux [FMP05, JF06]. De manière générale, nous avons constaté que l'adaptation des implémentations existantes était extrêmement fastidieuse. A titre d'indication, l'utilisation de notre librairie pour résoudre le problème de la découverte des clés minimales d'une relation a été faite en quelques heures.

Dans la suite de cette section, nous allons présenter quelques expérimentations réalisées à partir de notre librairie pour la découverte des motifs fréquents. Les composants nécessaires à la résolution de ce problème ont été implémentés, avec une version optimisée du test du prédicat. Les expérimentations ont été faites avec l'algorithme *Apriori* sur des jeux de données de FIMI [BZ03, BGZ04].

Nous avons comparé l'efficacité de notre programme avec deux autres implémentations d'*Apriori* spécialement dédiées à ce problème : l'implémentation de B. Goethals [Goea] et l'implémentation de C. Borgelt [Bor04]. La première est une implémentation classique de l'algorithme *Apriori*, la deuxième est la meilleure implémentation d'*Apriori* connue à ce jour. Elle est développée en C et est fortement optimisée. Soulignons que l'implémentation développée à partir de la librairie ne contient pas toutes les optimisations de ces deux autres programmes.

L'objectif de ces expérimentations est de montrer qu'une implémentation issue de la librairie permet le passage à l'échelle, et reste comparable aux meilleures implémentations, tout en étant générique et modulaire.

La figure 9.7 montre les temps d'exécution obtenus pour des seuils de support donnés (entre parenthèses sur la figure) pour les jeux de données *Connect*, *Pumsb* et *Pumsb*∗. L'implémentation issue de la librairie est celle notée "Apriori générique". Les résultats obtenus montrent que les performances d'"Apriori générique" se situent entre ceux des implémentations de B. Goethals et de C. Borgelt. Ces résultats sont donc très encourageants au regard de la rapidité et la simplicité d'obtention d'un programme opérationnel.

De plus, lors des différentes expérimentations réalisées, il est apparu que la majeure partie du temps d'exécution était liée à des composants développés (ou réutilisés) par l'utilisateur. La figure 9.8 montre la répartition du temps d'exécution d'*Apriori* développée à

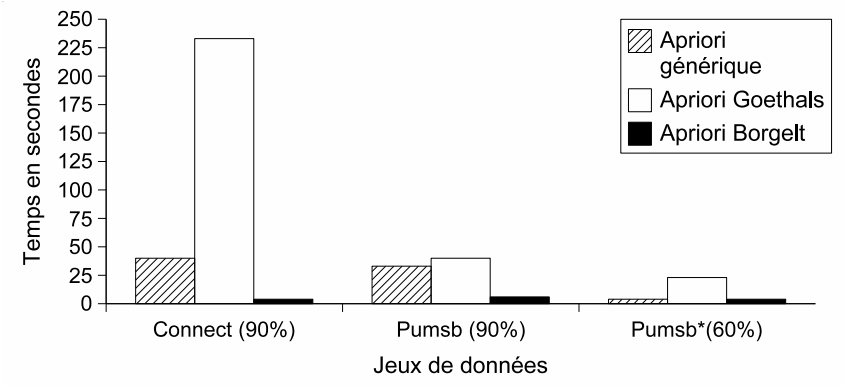

FIG. 9.7 – Expérimentation de trois implémentations d'Apriori sur trois jeux de données.

partir de la librairie pour le jeu de données *Connect* testé précédemment. Par conséquent, une implémentation plus optimisée de ces composants permettrait d'améliorer considérablement les performances globales de l'implémentation pour ce problème. Or, grâce à la modularité de la librairie, ces composants pourraient par la suite être directement réutilisés pour la résolution de nouveaux problèmes, permettant ainsi aux utilisateurs d'obtenir rapidement des solutions performantes.

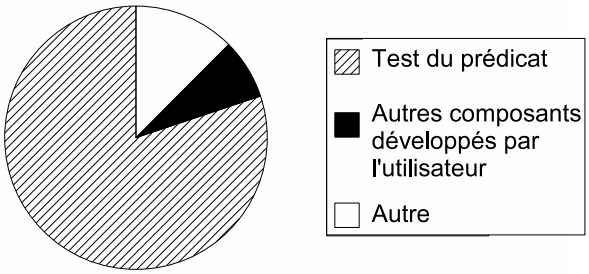

FIG. 9.8 – Répartition du temps d'exécution d'"Apriori générique" pour Connect (90%).

Nous avons étudié dans cette partie l'utilisation d'algorithmes d'extraction de motifs fréquents pour résoudre tout problème d'extraction de motifs intéressants, conforme au cadre théorique de [MT97]. Il est apparu que la majeure partie des algorithmes d'extraction de motifs fréquents étaient fortement liés à ce problème et pouvaient donc difficilement être appliqués à un autre. Dans ce contexte, nous avons présenté comment *ABS* s'appuyait sur une stratégie générique d'exploration de l'espace de recherche indépendante du prédicat,

dont l'objectif est de limiter le nombre de motifs testés. Nous avons aussi vu que ce type d'algorithme était notamment efficace pour résoudre le problème de la découverte des DI les plus satisfaites.

Ensuite, nous avons présenté un travail en cours permettant le prototypage rapide d'algorithmes d'extraction de motifs intéressants. Ce travail concerne de très nombreux problèmes mais avec un point de vue nouveau : celui de la généricité et du temps de développement nécessaire pour obtenir des programmes fiables, robustes et passant à l'échelle.

A notre connaissance, la seule contribution tendant vers les mêmes objectifs est DMTL. Quoique similaire dans l'esprit, notre proposition se démarque de DMTL en différents points :

- le cadre théorique est différent ;
- tout prédicat monotone ou anti-monotone peut être pris en compte alors que DMTL se focalise sur le prédicat "être fréquent" ;
- plus de stratégies de parcours disponibles. La librairie met notamment à disposition une version "générique" d'ABS permettant de trouver efficacement les bordures d'une théorie ;
- possibilité de découvrir la théorie ainsi que les bordures positive et négative.

Néanmoins, notre librairie est dans sa forme actuelle moins achevée que DMTL sur certains aspects tels que l'accès aux données.

Cinquième partie

Conclusion et perspectives

Conclusion

Depuis son introduction il y a plus de dix ans, la découverte de motifs fréquents, et plus généralement la découverte de motifs intéressants, a été un des problèmes de fouille de données les plus étudiés. Le nombre de problèmes de cette famille est important, et les domaines abordés variés. Toutefois, à notre connaissance, l'adaptativité et la généricité des algorithmes ont été peu étudiées menant à un grand nombre de ressources et de contributions hétérogènes. Afin d'unifier le paysage, des ateliers tels que FIMI [BZ03, BGZ04] ont eu pour objectifs de comparer et d'étudier le comportement des algorithmes de découverte des motifs fréquents, et surtout de rendre disponibles leurs implémentations.

Dans ce contexte, nous avons mis en oeuvre une approche où l'adaptativité et la généricité étaient des pré-requis, ce qui était clairement en rupture avec la tendance actuelle.

Tout d'abord, nous avons effectué une étude expérimentale approfondie des jeux de données utilisés pour l'extraction des motifs fréquents [FMP05]. A l'issue de ces expérimentations, il est apparu que la distribution de la bordure négative et de la bordure positive avait un impact important sur les algorithmes, et nous a mené à définir une nouvelle et meilleure classification des jeux de données.

Cette classification est plus simple que celle présentée dans [GZ01] tout en étant stable par rapport au changement de seuil minimum de support et en accord avec les performances des algorithmes. En plus des caractéristiques habituellement étudiées, la "distance" entre les distributions de la bordure négative et de la bordure positive permet de mieux évaluer la difficulté d'un jeu de données.

Par la suite, nous nous sommes appuyés sur ces résultats expérimentaux pour proposer un algorithme de découverte des bordures des motifs fréquents, appelé *ABS* (*Adaptive Borders Search*) [FMP04], adaptant dynamiquement sa stratégie d'exploration de l'espace de recherche en fonction des données. Cet algorithme s'appuie sur un parcours par niveau de l'espace de recherche de type *Apriori* [AS94], puis sur une alternance de "sauts" ou dualisations entre les deux bordures jusqu'à leur découverte.

La stratégie adaptative utilisée dans *ABS* consiste à déterminer dynamiquement à quel moment l'algorithme doit passer d'une exploration par niveau, à une approche alternant les dualisations. Ce choix est fait lorsque l'élagage fait par *Apriori* n'est plus efficace, i.e. lorsque la proportion de motifs candidats non fréquents générés est très faible. Dans ce cas, en exploitant les motifs des bordures déjà découverts, les dualisations permettent de faire de grands sauts dans l'espace de recherche, et évitent ainsi de tester un grand nombre de motifs. La stratégie adaptative exploite aussi une mesure d'erreur pour estimer la distance entre un motif non fréquent découvert par dualisation et la bordure positive. Les motifs identifiés comme étant "proches" de la bordure positive sont ensuite utilisés comme point de départ pour découvrir les motifs de la bordure positive sous-jacents.

Avec la généricité en tête, nous avons ensuite étudié les problèmes d'extraction de motifs intéressants, représentables par des ensembles, tels qu'ils ont été définis dans [MT97]. En effet, un des intérêts d'*ABS* est de s'appuyer sur une stratégie totalement générique,

dont l'efficacité est indépendante du prédicat étudié. Les sauts effectués ne sont pas fondés sur des heuristiques ou des propriétés liées au problème étudié, mais uniquement sur l'anti-monotonie du prédicat. Les performances de ce type d'approche ont notamment été mises en évidence pour le problème, complètement différent des fréquents, de la découverte de DI [MP03].

Nous nous sommes alors attaqués à la mise en place des algorithmes au niveau logiciel partant du constat que l'obtention d'une implémentation performante de ces algorithmes est longue et complexe, et reste confinée à la discrétion de quelques programmeurs spécialistes. Même l'adaptation des implémentations existantes s'avère difficile, et requiert une connaissance approfondie des techniques et optimisations mises en oeuvre. De plus, le code source, même libre et (parfois) commenté, est en général complexe et intimement lié au problème des motifs fréquents.

L'objectif était donc de faciliter le travail des programmeurs pour leur permettre d'obtenir un programme passant à l'échelle pour une application donnée. Dans ce contexte, nous avons proposé une librairie permettant le prototypage rapide de programmes d'extraction de motifs intéressants dans des données. Nous avons aussi proposé une méthodologie pour vérifier que le problème étudié correspond au cadre considéré dans ce mémoire, et pour guider l'utilisateur dans le développement des différents composants.

Cette librairie met à disposition de l'utilisateur des stratégies génériques telles qu'un parcours par niveau ou l'algorithme *ABS*, pour résoudre tout problème d'extraction de motifs intéressants reformulable dans le cadre théorique de [MT97]. Ces algorithmes permettent de découvrir la théorie, et/ou la bordure positive, et/ou la bordure négative. Les stratégies disponibles dans la librairie sont des boîtes noires pour les utilisateurs, qui doivent uniquement développer ou réutiliser les composants spécifiques au problème étudié.

Notre travail concerne donc la famille des problèmes d'extraction de motifs intéressants avec un point de vue nouveau : celui de la généricité et du temps de développement nécessaire pour obtenir des programmes fiables, robustes et passant à l'échelle. Nous avons mis en oeuvre cette approche pour résoudre des problèmes totalement différents de celui des fréquents tels que l'extraction de dépendances d'inclusion [MLP02, MP03] et la résolution de certaines phases du processus de réécriture de requêtes dans un système d'intégration de sources de données hétérogènes [JPR+05, JF06]. Les résultats obtenus ont montré qu'il était possible d'obtenir rapidement une implémentation utilisable, tout en permettant le passage à l'échelle. La librairie, encore en cours de développement, est librement accessible sur internet.

Perspectives

Les perspectives suscitées par ces travaux sont multiples. Les expérimentations réalisées sur les jeux de données ont permis d'étudier différents types de motifs (fréquents et représentations condensées des fréquents). Des propriétés non justifiées d'un point de vue

théorique ont ainsi été observées, telles qu'un grand nombre de distributions sous forme de "courbes en cloche", et la "stabilité" des distributions par rapport au changement de seuil minimum de support. Par la suite, notre étude s'est principalement focalisée sur les motifs fréquents et leur bordure. Bien qu'ayant obtenu une grande quantité d'informations sur les représentations condensées des fréquents, celles-ci ont été peu exploitées en pratique. Il serait donc particulièrement intéressant d'étudier en détail ces différents aspects. De nouvelles stratégies pourraient éventuellement naître.

Comme nous l'avons vu en section 5.3, cette étude serait aussi utile pour d'autres applications telle que la construction de jeux de données synthétiques plus réalistes. Des travaux tels que [QT, RMZ03, RZM05, DM05] ont proposé des méthodes pour générer des jeux de données synthétiques. Toutefois, au regard des expérimentations réalisées, il est apparu que ces méthodes ne permettaient pas de refléter la complexité et la diversité réelles des jeux de données.

Pour finir, nos travaux sur le prototypage rapide d'algorithmes d'extraction de motifs intéressants permettent d'envisager de nouvelles perspectives. L'ajout de nouveaux composants dans la librairie faciliterait son utilisation. La résolution de nouveaux problèmes se ferait plus rapidement et simplifierait le travail de l'utilisateur. Les composants à développer sont typiquement de nouvelles structures de données, ou des composants permettant de stocker et de gérer de nouveaux formats et sources de données. Il serait aussi intéressant d'inclure de nouveaux algorithmes à la librairie, tels que des algorithmes faisant un parcours en profondeur. Nous pouvons aussi essayer de relaxer certaines contraintes pour envisager de traiter plus de problèmes, comme la découverte de séquences fréquentes.

On peut imaginer le développement d'une approche plus déclarative de ces problèmes de fouilles de données en définissant un langage de requêtes qui, associé à un modèle de coût, permettrait à un optimiseur de choisir automatiquement le parcours à utiliser, libérant le développeur de cette tâche difficile.

Bibliographie

[AAP01] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent item sets. *J. Parallel Distrib. Comput.*, 61(3) :350–371, 2001.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *SIGMOD Conference*, pages 207–216. ACM Press, 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB*, pages 487–499. Morgan Kaufmann, 1994.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pages 3–14. IEEE Computer Society, 1995.

[Bay98] Roberto J. Bayardo Jr. Efficiently mining long patterns from databases. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD Conference*, pages 85–93. ACM Press, 1998.

[BBR03] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets : A condensed representation of boolean data for the approximation of frequency queries. *Data Min. Knowl. Discov.*, 7(1) :5–22, 2003.

[BCF+03] Douglas Burdick, Manuel Calimlim, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Mafia : A performance study of mining maximal frequent itemsets. In Bayardo Jr. and Zaki [BZ03].

[BCG01] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia : A maximal frequent itemset algorithm for transactional databases. In *ICDE*, pages 443–452. IEEE Computer Society, 2001.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BDFS84] Catriel Beeri, Martin Dowd, Ronald Fagin, and Richard Statman. On the structure of armstrong relations for functional dependencies. *J. ACM*, 31(1) :30–46, 1984.

[Ber73] Claude Berge. *Graphs and Hypergraphs*. North Holland, Amsterdam, 1973.

[BGZ04] Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[Bod03] Ferenc Bodon. A fast apriori implementation. In Bayardo Jr. and Zaki [BZ03].

[Bod04] Ferenc Bodon. Surprising results of trie-based fim algorithms. In Bayardo Jr. et al. [BGZ04].

[Bor03] Christian Borgelt. Efficient implementations of Apriori and Eclat. In Bayardo Jr. and Zaki [BZ03].

[Bor04] Christian Borgelt. Recursion pruning for the apriori algorithm. In Bayardo Jr. et al. [BGZ04].

[BR01] Artur Bykowski and Christophe Rigotti. A condensed representation to find frequent patterns. In *PODS*. ACM, 2001.

[BZ03] Roberto J. Bayardo Jr. and Mohammed Javeed Zaki, editors. *FIMI '03, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, November 19, 2003*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[CCL05] Alain Casali, Rosine Cicchetti, and Lotfi Lakhal. Essential patterns : A perfect cover of frequent patterns. In A. Min Tjoa and Juan Trujillo, editors, *DaWaK*, volume 3589 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2005.

[CG03] Toon Calders and Bart Goethals. Minimal k-free representations of frequent sets. In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, and Ljupco Todorovski, editors, *PKDD*, volume 2838 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 2003.

[DBL02] *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 2002.

[DM05] Didier Devaurs and Fabien De Marchi. Génération de bases de transactions synthétiques : vers la prise en compte des bordures. In Véronique Benzaken, editor, *BDA*, 2005.

[DT95] János Demetrovics and Vu Duc Thi. Some remarks on generating armstrong and inferring functional dependencies relation. *Acta Cybern.*, 12(2) :167–180, 1995.

[EG95] Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.*, 24(6) :1278–1304, 1995.

[FK96] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21(3) :618–628, 1996.

[Flo05] Frédéric Flouvat. Experimental study of frequent itemsets datasets. Technical report, LIMOS, France, http ://www.isima.fr/flouvat/papers/rr05-ExpStudyDatasets.pdf, june 2005.

[FMP04] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. ABS : Adaptive Borders Search of frequent itemsets. In Bayardo Jr. et al. [BGZ04].

[FMP05] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. A thorough experimental study of datasets for frequent itemsets. In *ICDM*, pages 162–169. IEEE Computer Society, 2005.

[FMP06] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. Vers le prototypage rapide de programmes de fouille de données. In Dominique Laurent, editor, *BDA*, 2006.

[FPSS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3) :37–54, 1996.

[GKM+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharm. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2) :140–174, 2003.

[GNZ05] Bart Goethals, Siegfried Nijssen, and Mohammed Javeed Zaki. Open source data mining : Workshop report. In *OSDM'05, Proceeding of the ACM SIGKDD Open Source Data Mining Workshop, Chicago,USA*, 2005.

[Goea] Bart Goethals. Apriori implementation. University of Antwerp. http ://www.adrem.ua.ac.be/~goethals/.

[Goeb] Bart Goethals. Frequent itemset mining implementations repository, http ://fimi.cs.helsinki.fi/.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*. Springer-verlag, 1999.

[GZ01] Karam Gouda and Mohammed Javeed Zaki. Efficiently mining maximal frequent itemsets. In Nick Cercone, Tsau Young Lin, and Xindong Wu, editors, *ICDM*, pages 163–170. IEEE Computer Society, 2001.

[GZ03] Gösta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In Bayardo Jr. and Zaki [BZ03].

[Hal01] Alon Y. Halevy. Answering queries using views : A survey. *VLDB J.*, 10(4) :270–294, 2001.

[HCS+05] Mohammad Hasan, Vineet Chaoji, Saeed Salem, Nagender Parimi, and Mohammed Zaki. DMTL : A generic data mining template library. In *Workshop on Library-Centric Software Design (LCSD'05), with Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'05) conference, San Diego, California*, 2005.

[HG05] Jiawei Han and Data Mining Group. IlliMine project. University of Illinois Urbana-Champaign Database and Information Systems Laboratory. http ://illimine.cs.uiuc.edu/, 2005.

[HMS01] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD Conference*, pages 1–12. ACM, 2000.

[IWM00] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In Djamel A. Zighed, Henryk Jan Komorowski, and Jan M. Zytkow, editors, *PKDD*, volume 1910 of *Lecture Notes in Computer Science*, pages 13–23. Springer, 2000.

[JF06] Hélène Jaudoin and Frédéric Flouvat. Techniques de fouille de données pour la réécriture de requêtes en présence de contraintes de valeurs. In Gilbert Ritschard and Chabane Djeraba, editors, *EGC*, volume RNTI-E-6 of *Revue des Nouvelles Technologies de l'Information*, pages 77–88. Cépaduès-Éditions, 2006.

[JPR+05] Hélène Jaudoin, Jean-Marc Petit, Christophe Rey, Michel Schneider, and Farouk Toumani. Query rewriting using views in presence of value constraints. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Description Logics*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[KG02] Marzena Kryszkiewicz and Marcin Gajek. Concise representation of frequent patterns based on generalized disjunction-free generators. In Ming-Shan Cheng, Philip S. Yu, and Bing Liu, editors, *PAKDD*, volume 2336 of *Lecture Notes in Computer Science*, pages 159–171. Springer, 2002.

[KMRS92] Martti Kantola, Heikki Mannila, Kari-Jouko Räihä, and Harri Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7 :591–607, 1992.

[KR03] Andreas Koeller and Elke A. Rundensteiner. Discovery of high-dimensional. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *ICDE*, pages 683–685. IEEE Computer Society, 2003.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.

[LK98] Dao-I Lin and Zvi M. Kedem. Pincer search : A new algorithm for discovering the maximum frequent set. In Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *EDBT*, volume 1377 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 1998.

[LL99] Mark Levene and George Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-verlag, 1999.

[LLY+03] Guimei Liu, Hongjun Lu, Jeffrey Xu Yu, Wang Wei, and Xiangye Xiao. Afopt : An efficient implementation of pattern growth approach. In Bayardo Jr. and Zaki [BZ03].

[LPL00] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and armstrong relations. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *EDBT*, volume 1777 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2000.

[LPT02] Stéphane Lopes, Jean-Marc Petit, and Farouk Toumani. Discovering interesting inclusion dependencies : application to logical database tuning. *Inf. Syst.*, 27(1) :1–19, 2002.

[MCP98] Florent Masseglia, Fabienne Cathala, and Pascal Poncelet. The psp approach for mining sequential patterns. In Jan M. Zytkow and Mohamed Quafafou, editors, *PKDD*, volume 1510 of *Lecture Notes in Computer Science*, pages 176–184. Springer, 1998.

[MFP05] Fabien De Marchi, Frédéric Flouvat, and Jean-Marc Petit. Adaptive strategies for mining the positive border of interesting patterns : Application to inclusion dependencies in databases. In Jean-François Boulicaut, Luc De Raedt, and Heikki Mannila, editors, *Constraint-Based Mining and Inductive Databases*, volume 3848 of *Lecture Notes in Computer Science*, pages 81–101. Springer, 2005.

[Mit83] John C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56(3) :154–173, 1983.

[MLP02] Fabien De Marchi, Stéphane Lopes, and Jean-Marc Petit. Efficient algorithms for mining inclusion dependencies. In Christian S. Jensen, Keith G. Jeffery, Jaroslav Pokorný, Simonas Saltenis, Elisa Bertino, Klemens Böhm, and Matthias Jarke, editors, *EDBT*, volume 2287 of *Lecture Notes in Computer Science*, pages 464–476. Springer, 2002.

[MP03] Fabien De Marchi and Jean-Marc Petit. Zigzag : a new algorithm for mining large inclusion dependencies in database. In *ICDM*, pages 27–34. IEEE Computer Society, 2003.

[MR94] Heikki Mannila and Kari-Jouko Räihä. *The Design of Relational Databases*. Addison-Wesley, second edition, 1994.

[MS00] Shinichi Morishita and Jun Sese. Traversing itemset lattice with statistical metric pruning. In *PODS*, pages 226–236. ACM, 2000.

[MT96] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations (extended abstract). In *KDD*, pages 189–194, 1996.

[MT97] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Min. Knowl. Discov.*, 1(3) :241–258, 1997.

[OLP+03] Salvatore Orlando, Claudio Lucchese, Paolo Palmerini, Raffaele Perego, and Fabrizio Silvestri. kdci : a multi-strategy algorithm for mining frequent sets. In Bayardo Jr. and Zaki [BZ03].

[OPPS02] Salvatore Orlando, Paolo Palmerini, Raffaele Perego, and Fabrizio Silvestri. Adaptive and resource-aware mining of frequent sets. In *ICDM* [DBL02], pages 338–345.

[PBTL99] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In Catriel Beeri and Peter Buneman, editors, *ICDT*, volume 1540 of *Lecture Notes in Computer Science*, pages 398–416. Springer, 1999.

[PHM00] Jian Pei, Jiawei Han, and Runying Mao. Closet : An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[POP04] Paolo Palmerini, Salvatore Orlando, and Raffaele Perego. Statistical properties of transactional databases. In Hisham Haddad, Andrea Omicini, Roger L. Wainwright, and Lorie M. Liebrock, editors, *SAC*, pages 515–519. ACM, 2004.

[QT] The Quest Team. Synthetic data generation code for associations and sequential patterns. Intelligent information systems, IBM almaden research center. http ://www.almaden.ibm.com/software/quest/Resources/.

[RMZ03] Ganesh Ramesh, William Maniatty, and Mohammed Javeed Zaki. Feasible itemset distributions in data mining : theory and application. In *PODS*, pages 284–295. ACM, 2003.

[Rym92] Ron Rymon. Search through systematic set enumeration. In *KR*, pages 539–550, 1992.

[RZM05] Ganesh Ramesh, Mohammed Javeed Zaki, and William Maniatty. Distribution-based synthetic database generation techniques for itemset mining. In *IDEAS*, pages 307–316. IEEE Computer Society, 2005.

[SU03] Ken Satoh and Takeaki Uno. Enumerating maximal frequent sets using irredundant dualization. In Gunter Grieser, Yuzuru Tanaka, and Akihiro Yamamoto, editors, *Discovery Science*, volume 2843 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 2003.

[TRS02] Alexandre Termier, Marie-Christine Rousset, and Michèle Sebag. Treefinder : a first step towards xml data mining. In *ICDM* [DBL02], pages 450–457.

[UAUA03] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. Lcm : An efficient algorithm for enumerating frequent closed item sets. In Bayardo Jr. and Zaki [BZ03].

[UAUA04] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In Einoshin Suzuki and Setsuo Arikawa, editors, *Discovery Science*, volume 3245 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[UKA04] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2 : Efficient mining algorithms for frequent/closed/maximal itemsets. In Bayardo Jr. et al. [BGZ04].

[Ull00] Jeffrey D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2) :189–210, 2000.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining : Practical machine learning tools and techniques, 2nd Edition.* Morgan Kaufmann, San Francisco, 2005.

[Wil82] Rudolf Wille. *Restructuring lattice theory : An approach based on hierarchies of concepts*, pages 445–470. Ordered Sets. Reidel,Dordrecht-Boston, 1982.

[Zak00] Mohammed Javeed Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(2) :372–390, 2000.

[Zak02] Mohammed Javeed Zaki. Efficiently mining frequent trees in a forest. In *KDD*, pages 71–80. ACM, 2002.

[ZH02] Mohammed Javeed Zaki and Ching-Jiu Hsiao. Charm : An efficient algorithm for closed itemset mining. In Robert L. Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila, and Rajeev Motwani, editors, *SDM*. SIAM, 2002.

[ZPD+05] Mohammed Javeed Zaki, Nagender Parimi, Nilanjana De, Feng Gao, Benjarath Phoophakdee, Joe Urban, Vineet Chaoji, Mohammad Al Hasan, and Saeed Salem. Towards generic pattern mining. In Bernhard Ganter and Robert Godin, editors, *ICFCA*, volume 3403 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2005.

[ZPOL97] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *KDD*, pages 283–286, 1997.